Published in: Parallel Processing Letters, Online Ready, May 9, 2025
DOI: https://doi.org/10.1142/S0129626425500045

Computing and communication structure design for fast mass-parallel numerical solving PDE

Dmitry A. Zaitsev[1], Tatiana R. Shmeleva[2], Alexander A. Kostikov[3]

[1] The University of Derby, Derby, UK, ORCID: 0000-0001-5698-7324
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany, ORCID: 0000-0002-4799-3842
[3] Technical University "Metinvest Polytechnic" LLC, Zaporizhzhia, Ukraine, ORCID: 0000-0003-3503-4836

daze@acm.org

Abstract. Partial differential equations and systems with certain boundary conditions specify continuous processes significant for both large-scale simulations in computer-aided design using HPC and subsequent real-time control of embedded applications using dedicated hardware. The paper develops a spectrum of techniques based on a family of place-transition nets aimed at the computing and communication structure design for fast mass-parallel numerical solving of PDEs. For the HPC domain, we develop models of interconnects in the form of infinite nets and graphical programs in the form of Sleptsov nets. For the embedded control domain, we develop specialized lattices for fast numerical solving PDE based on integer number approximation specified with Sleptsov-Salwicki nets to be implemented on dedicated hardware, which we prototype on FPGAs. For mass-parallel solving of PDEs, we employ ad-hoc finite-difference schemes and iteration methods that allow us to recalculate the lattice values in a single time cycle suitable for control of hypersonic objects and thermonuclear reactions.

Keywords: Partial-differential equation, finite difference, iteration, computing lattice, mass-parallel solution, embedded control, FPGA

1 Introduction

Solving a Partial Differential Equation (PDE) [1-3] represents the basic technique of continuous system modeling in a wide range of practical applications. Some specific forms of PDEs allow us to obtain analytical solutions, which simplify the process of their application. In the general case, a given equation is resolved numerically; either finite difference or finite element techniques are applied. A Computer Aided Design (CAD) [4] represents the basic application area. For big, realistic tasks, high-performance computing (HPC) [5] resources are involved. We take this HPC domain into consideration, focusing also on specific requirements and techniques for fast solving of PDEs for control purposes in embedded applications [6]. For example, control of airplane engine turbines based on resolving gas dynamics and thermodynamics problems, control of hypersonic vehicle trajectories, etc.

For these applications, iteration methods [2] yield the utmost performance, allowing us to apply directly a mass-parallel computing approach. When solving PDE numerically, via the iteration technique, based on the finite difference approximation [1-3] to achieve maximal performance, a computing structure should be “isomorphic” to the task space. Thus, a

multidimensional lattice of computing nodes is the most suitable structure. In this paper, we distinguish between general-purpose computing lattices and specialized structures to solve fast a single class of PDE appropriate for a chosen embedded application. We apply a general-purpose computing lattice with definite spatial structure, inspired by the Tofu D interconnect approach [7], which connects nodes of Fugaku [8] supercomputer using a 6D torus. The choice is motivated by the fact that Fugaku represents the most efficient supercomputer [9]. In essence, Fugaku embodies an early approach to specialized supercomputer design dedicated to resolving quantum physics tasks [10]. For embedded applications, we offer an approach that implements a specialized hardware lattice on FPGA [11] via generating the corresponding Verilog [12] code.

For specification of computing and communication lattices, we use a family of languages based on Sleptsov nets [13] and their extensions with infinite [14,15] and loaded (colored) [16,17] nets and a special class of reenterable models [18] which represents an abstraction convenient for lattice specification prior to Verilog code synthesis.

We model an HPC interconnect as a multidimensional cube of nodes [19] using real-life boundary conditions [20] to specify the packet delivery between customer (terminal) devices connected to the boundary of the communication matrix; otherwise, we close (connect) the opposite bounds to obtain the torus surface of computing-communication nodes. We study such basic principles of packet switching as (a) store-and-forward, (b) cut-through, and (c) combined. Such well-known formalisms of mathematical modeling as cellular automata [21,22] and Petri nets have close intrinsic relations, especially Petri nets with a regular spatial structure, for instance, square. We apply an early-introduced formalism of infinite Petri nets [14,15] for specifying communication lattice models composed in the present paper.

For embedded applications, we considerably simplify computing-communication structure based on integer number approximation converging with cellular automata techniques for dedicated hardware implementation in the form of a computing lattice, which we prototype on FPGA [11]. Obtained for FPGA benchmarks acknowledge the applicability of the approach for fast control based on the ongoing process of numerically solving boundary problems for PDE, where the boundary is actually mapped into sensors and actuators.

In Section 2, we study the iteration techniques for the finite difference method of solving PDE and focus on a stencil with Moore's neighborhood that yields good convergence with a fourth degree of accuracy with respect to the mesh step. In Section 3, we study computational aspects of the node work and specify the node software by a graphical concurrent Sleptsov net program using colored nets. In Section 4, we study packet switching communication lattices with Moore's neighborhood and modern switching techniques, specifying them by infinite Petri nets, reducing tasks of their analysis to solving infinite Diophantine systems of equations. In Section 5, we focus on the peculiarities of dedicated computing-communication lattice design for embedded applications composing reenterable colored net models as an abstract specification. Then we generate Verilog code for prototyping specialized lattices, which solve PDE on FPGA, based on their specification by the plain Sleptsov-Salwicki nets. Obtained benchmarks prove fast convergence and high accuracy of the chosen finite difference scheme, acknowledging a rather good integer approximation for embedded applications.

2 Iteration Finite Difference Method for Solving PDE

Partial differential equations are widely used to describe physical phenomena in such domains as mechanics, hydrodynamics, acoustics, heat transfer, electricity, magnetism, and others. In most cases, it is impossible to find analytically the solution of these equations. Therefore, numerical methods, in particular, the finite difference method, have become widespread for their solution [1-3].

In the finite difference method, the original partial derivative equation is reduced to a system of difference equations, for the solution of which, direct and iterative methods are applied. Often, this is the only way to find the solution of partial derivative equations.

The essence of the finite difference method is to solve difference equations that are obtained from the original partial derivative equation by replacing the partial derivatives by their difference analogs with a greater or lesser degree of accuracy.

For example, applying the finite difference method, the Laplace equation

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = 0$$

can be replaced by the following difference relation

$$\frac{u(x-h,y) - 2u(x,y) + u(x+h,y)}{h^2} + \frac{u(x,y-h) - 2u(x,y) + u(x,y+h)}{h^2} = 0,$$

where $h > 0$ is the discretization step.

With this substitution, we arrive at an equation that relates the values of the desired function at separately taken points, which are usually chosen so that they form a square mesh.

Let us consider different ways of approximating partial derivatives and their corresponding patterns, which are most often encountered in solving problems of mathematical physics. We construct the difference approximations for the case of a function of two independent variables $u(x,y)$ on a rectangular mesh. Without restricting the generality, we will assume that the change area of the argument x is the segment $0 \le x \le 1$, and the change area of the argument y is the segment $0 \le y \le 1$. Let us divide the segments $0 \le x \le 1$, $0 \le y \le 1$ by points $x_i = ih$ $(i = 0,1,2 \dots, N)$, $y_j = jh$ $(j = 0,1,2, \dots, N)$ into N equal parts of length $h = \frac{1}{N}$ each. The set of points (x_i, y_j) with coordinates $x_i = ih$ $(i = 0,1,2 \dots, N)$ and $y_j = jh$ $(j = 0,1,2, \dots, N)$ is called a mesh in the square $0 \le x \le 1$, $0 \le y \le 1$. By $u_{i,j}$ we denote the value of function u at the points with coordinates (x_i, y_j): $u_{i,j} = u(x_i, y_j)$.

Let us give some difference analogs of partial derivatives, which are used when applying the finite difference method. The following finite-difference approximations can be found in textbooks [1-3]. Fig. 1 shows stencils corresponding to the given difference operators.

1. $\frac{\partial u_{i,j}}{\partial x} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2h}$, error order $O(h^2)$ (Fig. 1a).

2. $\frac{\partial u_{i,j}}{\partial x} \approx \frac{u_{i+1,j+1} - u_{i-1,j+1} + u_{i+1,j-1} + u_{i-1,j-1}}{2h}$, error order $O(h^4)$ (Fig. 1b).

3. $\frac{\partial u_{i,j}}{\partial y} \approx \frac{u_{i,j+1}-u_{i,j-1}}{2h}$, error order $O(h^2)$ (Fig. 1c).

4. $\frac{\partial u_{i,j}}{\partial y} \approx \frac{u_{i+1,j+1}-u_{i+1,j-1}+u_{i-1,j+1}+u_{i-1,j-1}}{4h}$ error order $O(h^4)$ (Fig. 1b).

5. $\frac{\partial^2 u_{i,j}}{\partial x^2} \approx \frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$, error order $O(h^2)$ (Fig. 1d).

6. $\frac{\partial^2 u_{i,j}}{\partial x^2} \approx \frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}+u_{i+1,j}-2u_{i,j}+u_{i-1,j}+u_{i+1,j-1}-2u_{i,j-1}+u_{i-1,j-1}}{3h^2}$,

error order $O(h^4)$ (Fig. 1e).

7. $\frac{\partial^2 u_{i,j}}{\partial y^2} \approx \frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h^2}$, error order $O(h^2)$ (Fig. 1f).

8. $\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i+1,j+1}-2u_{i+1,j}+u_{i+1,j-1}+u_{i,j+1}-2u_{i,j}+u_{i,j-1}+u_{i-1,j+1}-2u_{i,j-1}+u_{i-1,j-1}}{3h^2}$

error order $O(h^4)$ (Fig. 1e).

9. $\frac{\partial^2 u_{i,j}}{\partial x \partial y} = \frac{u_{i+1,j+1}-u_{i+1,j-1}-u_{i-1,j+1}+u_{i-1,j-1}}{4h^2}$ error order $O(h^4)$ (Fig. 1b).

10. $\frac{\partial^4 u_{i,j}}{\partial x^4} = \frac{u_{i+1,j+1}+u_{i-1,j+1}+u_{i+1,k-1}+u_{i-1,j-1}-2u_{i+1,j}-2u_{i-1,j}-2u_{i,k+1}-2u_{i,k-1}+4u_{i,j}}{h^4}$

error order $O(h^4)$ (Fig. 1e).

Fig. 1 shows that the difference analogs 6, 8, and 10 form a Moore neighborhood [21,23]. Thus, the Moore neighborhood arises in the difference approximation of partial derivative equations containing derivatives of order 2 and 4. As an example of the application of the approximation leading to the Moore neighborhood, consider the Dirichlet problem for the Laplace equation.

We will solve the following problem: we need to find a function of two variables *u(x,y)*, satisfying in the domain *D={0<x<1,0<y<1}* to the Laplace equation

1. $$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = 0 \qquad (1)$$

and taking on the boundary ∂D of the domain *D* given values:

2. $$u|_{\partial D} = \varphi(x,y), \qquad (2)$$

where $\varphi(x,y)$ is a given function.

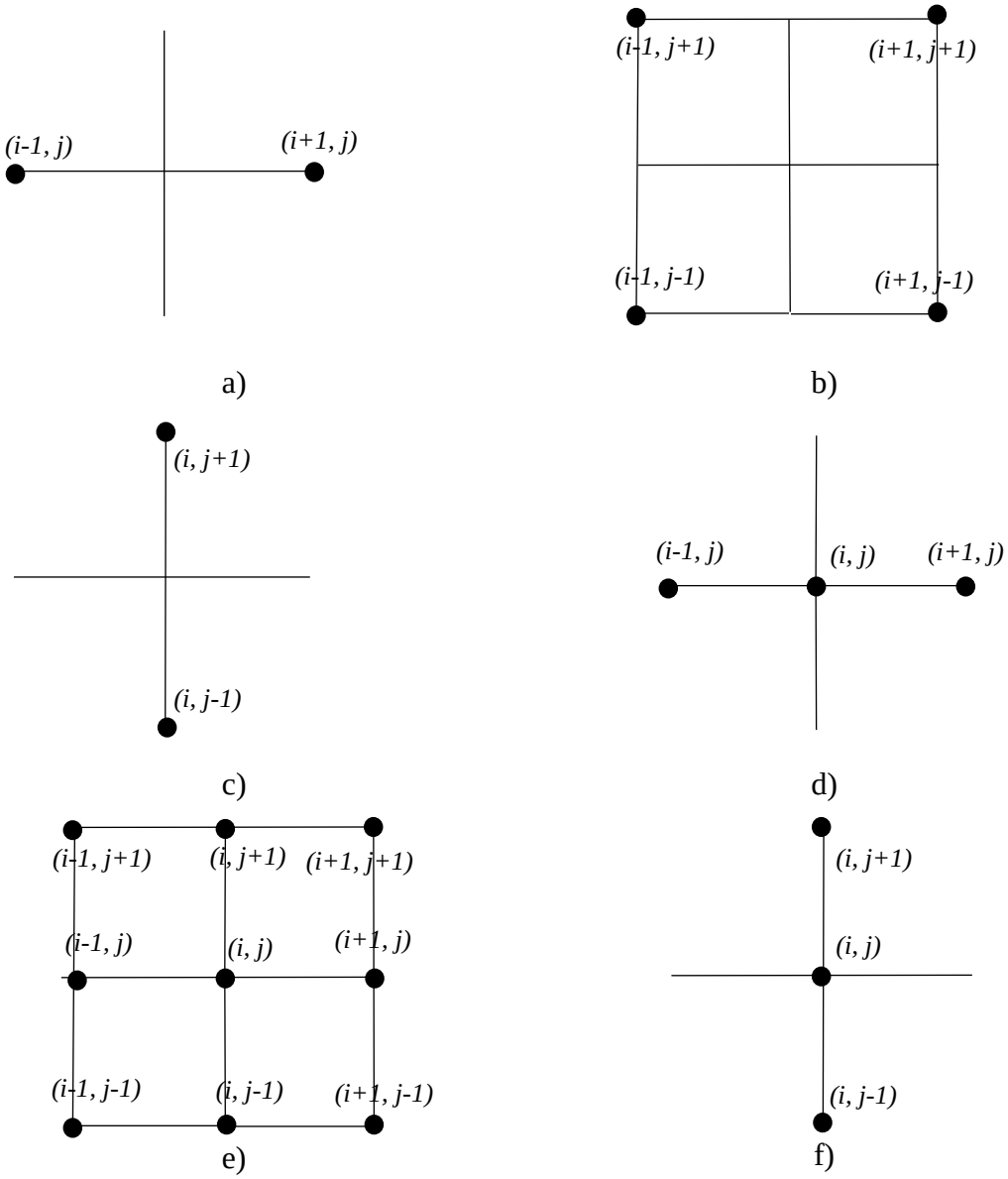


Fig. 1. Interior stencils for the finite difference method on a plane.

In domain *D*, we construct a mesh with step *h*, as described earlier. In [2], the following difference scheme is constructed to approximate the Laplace equation of 4th accuracy level $O(h^4)$:

$$u_{i,j} = \frac{4\left(u_{i,j-1} + u_{i-1,j} + u_{i,j+1} + u_{i+1,j}\right) + u_{i-1,j-1} + u_{i+1,j+1} + u_{i-1,j+1} + u_{i+1,j-1}}{20}$$

$$i = 1,2,\dots,N-1, j = 1,2,\dots,N-1 \qquad (3)$$

Here *N* is the number of partition points along the *x* and *y* directions.

The value of the sought function at the boundary nodes is determined from condition (2)

$$u_{i,0} = \varphi(x_i, 0) \qquad (4)$$

$$u_{i,N} = \varphi(x_i, 1) \qquad (5)$$

$$u_{0,j} = \varphi(0, y_j) \qquad (6)$$

$$u_{N,j} = \varphi(1, y_j) \qquad (7)$$

$$(i = 0,1,2, \dots, N, j = 1,2, \dots N-1).$$

We will use the method of simple iteration to solve the system of grid equations (3)-(7). Let's set the initial approximation $u_{i,j}^{(0)}$. For internal points of the domain *D*, we choose the initial approximation arbitrarily; at the boundary, the initial approximation is determined by expressions (4)-(7). The next approximation is determined by the formula

$$u_{i,j}^{(s+1)} = \frac{4\left(u_{i,j-1}^{(s)} + u_{i-1,j}^{(s)} + u_{i,j+1}^{(s)} + u_{i+1,j}^{(s)}\right) + u_{i-1,j-1}^{(s)} + u_{i+1,j+1}^{(s)} + u_{i-1,j-1}^{(s)} + u_{i+1,j-1}^{(s)}}{20}, \qquad (8)$$

where *s* is the iteration number *s*=0, 1, 2, …, $i = 1, 2, \dots, N-1, j = 1, 2, \dots, N-1$.

The calculations continue until the condition (9) is satisfied.

$$\max_{i,j} \left| u_{i,j}^{(s+1)} - u_{i,j}^{(s)} \right| < \varepsilon. \qquad (9)$$

In [2], it was shown that the solution of a problem (3)-(7) converges to the solution of a problem (1)-(2) at the rate of $O(h^4)$. It follows from the results of [2] that the iterative process (8)-(9) converges at any initial approximation. Since the connection between ε and h is rather sophisticated, we represent it in a simplified algorithm-like form. Given an accuracy of approximation ε, we compute the mesh step as $h = \sqrt[4]{\varepsilon}$ and the mesh size as $N = \lfloor 1/h \rfloor$, and run the iteration process (8)-(9) for initial values coinciding with some boundary value.

To perform the numerical experiment using the difference scheme (3)-(7), the solution of the temperature field distribution problem in a bar of square cross-section with unit length sides was found, assuming that the distribution along its faces is known. The solution of the system of linear equations (3)-(7) was found by the method of simple iteration (8)-(9). The problem was to find the solution of equation (1) in a square with sides of unit length and the following boundary conditions:

$$u(x, 0) = 1, \qquad 0 \le x \le 1$$

$$u(0, y) = 0, \qquad 0 < y \le 1$$

$$u(x, 1) = 0, \qquad 0 \le x < 1$$

$$u(1, y) = 1, \qquad 0 \le y \le 1$$

The results of calculations are shown in Fig. 2, from where it can be concluded that the steady-state temperature distribution inside the bar takes values between the minimum and maximum temperature values at the boundary, which agrees with the maximum principle satisfied by the solution of Laplace's equation.

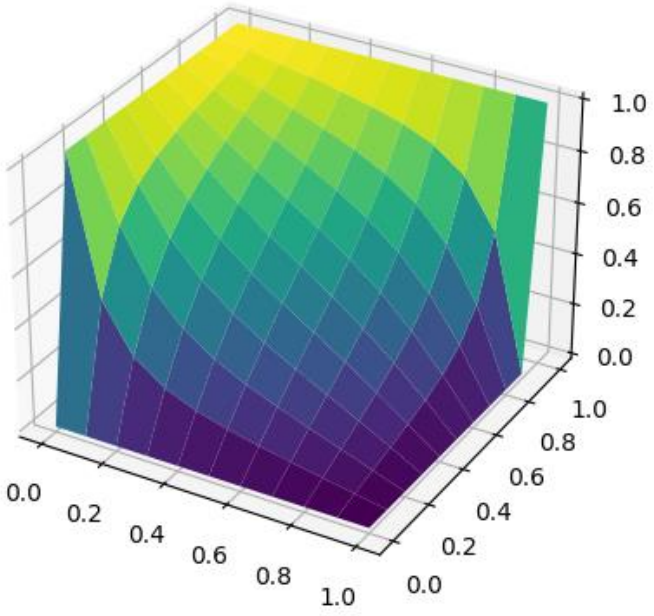


Fig. 2. The surface plot of the temperature distribution inside the beam cross-section.

This algorithm is simple to parallelize. If there is a sufficient number of processors, each iteration can be performed in one step, which will significantly increase the speed of finding a solution to the problem.

The solution of boundary value problems for the Laplace equation is of great practical importance. The problems of finding a stationary thermal field are reduced to the solution of this problem. For example, a constant temperature regime is given on the boundary of a domain, and it is necessary to find the temperature distribution inside the region. Likewise, the problems of finding the electric field potential and electrostatic field potential of the stationary current are reduced to the solution of the boundary value problem for the Laplace equation.

Let us consider some papers devoted to the solution of the Dirichlet problem for the Laplace equation. In [24], the distribution of electric potential in the region of interest was found by solving the Dirichlet problem for the Laplace equation with a given potential on the boundary. The authors of [24] conducted a comparative analysis of three iterative methods that were used to solve the corresponding difference problem: the Jacobi, the Gauss-Seidel, and Successive Over Relaxation (SOR) methods. It is shown that the SOR method is the most

efficient in terms of convergence rate. In [25], three methods were used to solve the Dirichlet problem for the Laplace equation: the finite difference method (FDM), the finite element method (FEM), and the Markov chain method (MoM). The obtained results were compared with the exact solution to verify the accuracy of the methods used. An accuracy investigation of three methods—the finite difference method (FDM), the finite element method (FEM), and the method of moments (MoM) to solve the Dirichlet problem for the Laplace equation—was conducted in [26]; advantages and disadvantages of these methods have been discussed.

3 Model of Computations within a Node

In this section, we develop a model of computations, which are implemented within a combined communication and computing node. In Section 4, we develop a model of communication lattice implemented in the hardware form of connected nodes based on a certain type of interconnect [7,27], which allows us to organize information exchange according to a given pattern of nodes interconnection in a specific finite difference scheme (stencil).

A bipartite graph, Gill [28] was using for modeling parallel computations, has been supplied by Carl Petri [29] with dynamic elements, called tokens, situated within vertices of the graph's first part, called places, and depicted as circles or ovals. Vertices of the second part, called transitions, and depicted as squares or rectangles, fire, consuming tokens from their input places and producing tokens within their output places. The discrete time process of firing transitions represents the Petri net (PN) behavior. Conventionally, finite sets of places and transitions have been considered [30]. A Petri net [30] represents a special case of a Sleptsov net (SN) [13] where a transition fires in multiple instances at a step. Sleptsov Net Computing (SNC) [31-33] resolves many problems of modern HPC. In this paper, we use SNC as the basic approach to represent computations; we suppose that nodes of a cluster (lattice) are supplied with SN machines [34] to run programs drawn in the SN graphical language [31].

The computation model is uploaded to the node’s SN machine for fast mass-parallel execution. Here we prefer using high-level loaded (colored) nets [16-17], where we manipulate by real numbers, supposing their implementation on low-level SNs studied in [34,35], where an alternative technique of solving PDE, reduced to diagonal linear systems, via the sweep method was presented for the case when the iterative approach is difficult to apply because of convergence issues.

In [36], the basics of modeling computer networks by Colored Petri Nets (CPNs) [17] are studied. The model represents a combination of a bipartite directed graph and Milner's functional programming Markup Language (ML) [37], which expressions inscribe the graph elements; separate declarations of data types (color sets), variables, and functions are provided. A CPN token represents an element of abstract data type. Attributes of a place include its name, data type, and initial marking. Attributes of a transition contain its name, timed delay, guard function, and action procedure. Inscriptions of a transition incoming arc represent a pattern for selecting tokens, while inscriptions of a transition outgoing arc represent constructors of resulting tokens. The model offers certain flexibility using arc and node inscriptions. For instance, the incoming tokens can be selected with the incoming arc

inscriptions and the transition guard function; the outgoing tokens can be produced within the transition action procedure and within inscriptions of the transition outgoing arcs.

The specification of the iteration scheme (8) by a loaded SN is shown in Fig. 3. For receiving values from the neighbors according to Moore's neighborhood [21,23], we have 8 ports whose names are specified in TEX-like notation; for example, "u_i-1,j+1" means $u_{i-1,j+1}$. For sending the current node value, we have 8 transitions "send to", which actually forward the message to the communication lattice studied in Section 4. The message is delivered to a neighboring node having the same algorithm as specified in Fig. 3 with a specific index. The message content is represented by the current node value "ij" corresponding to $u_{i,j}$ though it could be supplemented with other useful information, for instance, the current error computed according to (9). Note that the error (accuracy) computation process supposes a kind of synchronization involving all the nodes of a lattice that can considerably affect the model performance. In some cases, a preliminary evaluation of the iteration number, based on a given accuracy, can be useful as a simple to implement halt condition. We consider, as a future research direction, an asynchronous approach of stopping the node program when its local accuracy in the corresponding mesh point value approximation has been achieved.

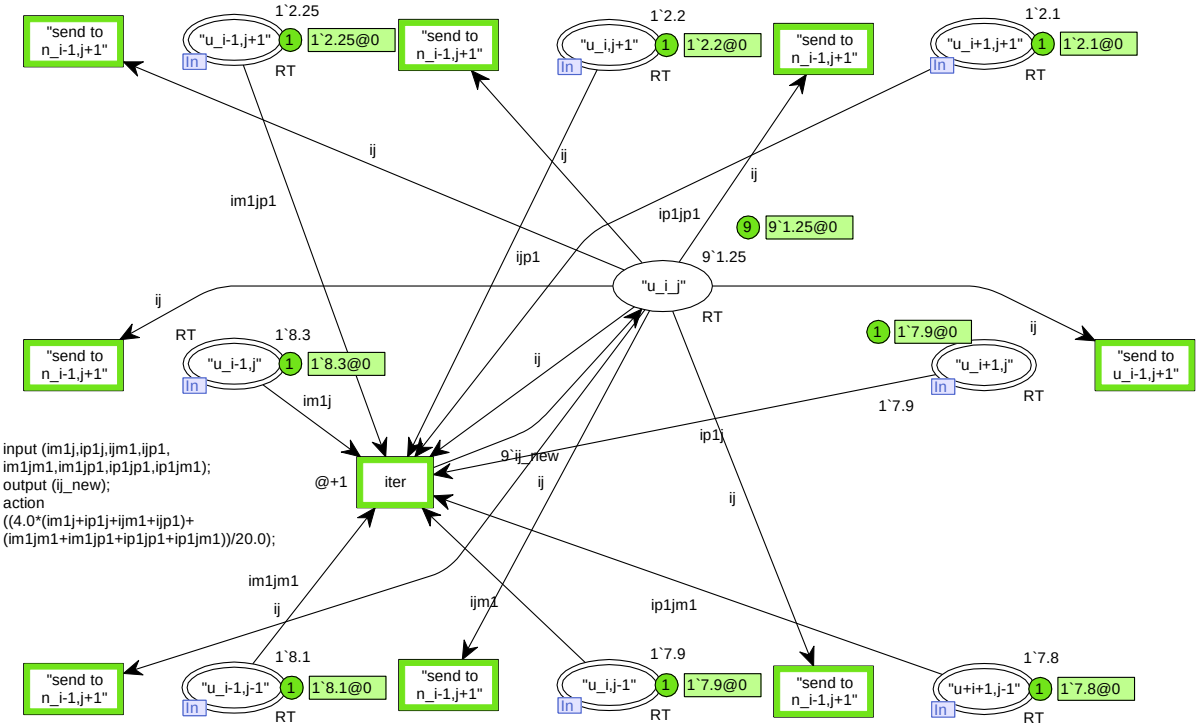


Fig. 3. SN program for solving PDE (1) on a lattice with Moore's neighborhood via finite difference scheme (8).

Actually, computing nodes, shown in Fig. 3 are connected via a communication network specified with a lattice (11)-(12) or (18)-(19) studied in Section 4. Because of this reason, we provide input places (ports) to input information and transitions to transmit the node output in the form of the recomputed magnitude of the current node. Thus, we can compose an explicit

abstract lattice, connecting the sending transitions with the corresponding port places of the neighboring nodes. We use a rather modest number of declarations, which follow:

```
colset RT= REAL timed;
var ij,ij_new,im1j,ip1j,ijm1,ijp1,im1jm1,im1jp1,ip1jp1,ip1jm1: RT;
```

The recalculation of the node magnitude according to (8) is represented by the action procedure of the transition *iter*, which specifies input and output values and the computation formula. If required, the formula can be represented completely in the SN graphical language as we expand it in the following sections. We use special notation for variables that extract tokens from neighboring nodes, omitting the function name and listing indices using character “m” instead of a minus (subtraction) operation and character “p” instead of a plus (addition) operation; thus, the value of "u_i-1,j+1" is extracted with the variable called *im1jp1*.

For the completion condition observation, we are required to arrange the computation of error to evaluate its maximal value, which can be implemented in a similar way within a node. The completion condition test for all the nodes can be arranged using the consensus procedures studied in [39].

4 General Purpose Communication Lattice Model with Moore’s Neighborhood

We compose a communication lattice of general form for numerical solving of PDEs via the finite difference method. The lattice composition in the form of a square with the Moore neighborhood [21,23] allows us to map various finite difference stencils for fast mass-parallel solving of PDEs.

In this section, models are represented in the form of an Infinite Petri net (IPN) [14,15]. Infinite Petri nets have been introduced and studied [14,15,19] for modeling complex networking protocols. For finite specification of IPNs, parametric multiset rewriting systems [39] are used. For IPNs analysis, we compose and solve infinite systems of linear Diophantine equations [40].

4.1 Model of communication device with cut-through switching mode

The device model for cut-through switching mode is shown in Fig. 4 in the form of a Petri net; the device ports are situated on the sides and corners of a unit-size square model. Ports are enumerated in two ways: ports 1-4 are clockwise starting from the upper port as sides of a square; port 5 is on the left upper corner of the square; port 6 is on the right upper corner; port 7 is on the right down corner; and port 8 is on the left down corner. The model contains eight ports with respect to Moore’s neighborhood [21,23]; each port is subdivided into the input and output channels (tracts). For the packet transmitting, the simple switching rules [41] are applied:

– a packet is received by an input port of the switching node;

– as the next hop forwarding destination, a random output port, among seven output ports (except the current input port), is chosen;

– the cut-through switching procedure forwards the packet to the output port directly without buffering.

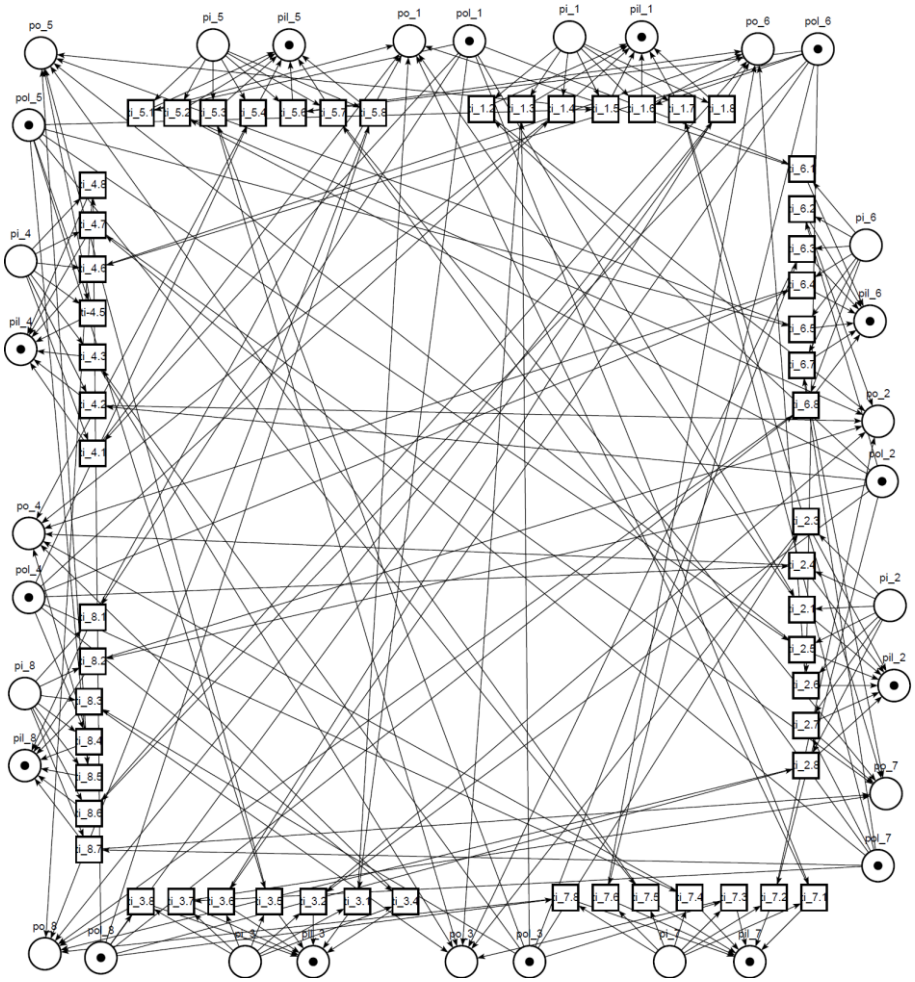


Fig. 4. Model of 8-ports communication device with cut-through switching.

An expression (10) represents a parametric description of the eight-port communication device model shown in Fig. 4. We use two parameters: the port number to which the packet is switched, denoted by the letter *v*; the port numbers of a device *u*. The parametric expression (10) contains only one line, which describes all transitions of the switch model, the input and the output tracts of all ports. Also, we use conventions [14,15] within the model notations, where the letter "t" denotes a transition and the letter "p" specifies a place. Transition $ti_{u,v}$ implements the packet forwarding decision, directly moving a packet from the current source port *u* to the destination output port *v*.

$$(((ti_{u,v} : pi_u, pol_v \rightarrow pil_u, po_v),\ v = \overline{1,8},\ v \neq u),\ u = \overline{1,8}) \qquad (10)$$

The letters "i" and "o" denote input and output tracts of the model. Place pi_u specifies the input port buffer of a unit size and place pil_u specifies the input port buffer size limitation having one token. Places po_v and pol_v specify the output port buffer and limitation of its size, measured in the number of packets, respectively.

Table 1 evaluates the model size of a switching device with the cut-through switching procedure measured in the number of places, transitions, and arcs.

Table 1. The switching device model size.

Number of places	Number of transitions	Number of arcs
(2+2)·8=	7·8=	(2+2)·7·8=
32	56	224

We use magnitudes from Table 1 in subsection 4.2 for the calculation of the total number of places, transitions, and arcs within the square lattice model having an arbitrary size *k.*

4.2 Specification of an open lattice with cut-through switching nodes

We use a direct parametric expression (11) of the device model with cut-through switching mode for the composition of an open communication lattice model [14] with an arbitrary size *k* in the form of an IPN. An open communication lattice means that there are no terminal (customer) devices, which are attached to the lattice border ports; otherwise, we call it a closed communication lattice. The specification of an open square lattice model with regular structure is given by (11) and (12).

$$\left(\left(\left(ti_{u,v}^{i,j}: pi_u^{i,j}, pol_v^{i,j} \rightarrow po_v^{i,j}, pil_u^{i,j}\right), \begin{matrix} 1 \le v \le 8, \\ v \ne u \end{matrix}\right), 1 \le u \le 8\right), \begin{matrix} 1 \le i \le k, \\ 1 \le j \le k \end{matrix}\right) \quad (11)$$

We added upper indexes *(i, j)* to the parametric description of the communication device model (10) with Moore's neighborhood, where index *i* is the row number and index *j* is the column number of a device within the lattice. The expression (11) specifies all nodes of a square lattice of size *k*×*k* while the expression (12) specifies all connections between the lattice nodes.

We consider separately eight types of connections between nodes within the lattice, in accordance with Moore's neighborhood, represented by eight corresponding lines of the parametric expression (11). Lines 1 and 5 describe horizontal connections; lines 2 and 6 describe vertical connections; lines 3 and 7 describe the main diagonal connections, and lines 4 and 8 describe the secondary diagonal connections.

$$\left(\begin{array}{l}
\left(ti_{i,j-1_i,j}: po_4^{i,j}, pil_2^{i,j-1} \rightarrow pi_2^{i,j-1}, pol_4^{i,j}\right), i = \overline{1,k}, j = \overline{2,k}, \\
\quad \left(ti_{i-1,j_i\ \ ,j}: po_1^{i,j}, pil_3^{i-1,j} \rightarrow pi_3^{i-1,j}, pol_1^{i,j}\right), i = \overline{2,k}, j = \overline{1,k}, \\
\left(ti_{i-1,j-1_{,i},j}: po_5^{i,j}, pil_7^{i-1,j-1} \rightarrow pi_7^{i-1,j-1}, pol_5^{i,j}\right), i = \overline{2,k}, j = \overline{2,k}, \\
\left(ti_{i,j-1_i-1,j}: po_8^{i-1,j}, pil_6^{i,j-1} \rightarrow pi_6^{i,j-1}, pol_8^{i-1,j}\right), i = \overline{2,k}, j = \overline{2,k}, \\
\left(to_{i,j-1_i,j}: po_2^{i,j-1}, pil_4^{i,j} \rightarrow pi_4^{i,j}, pol_2^{i,j-1}\right), i = \overline{1,k}, j = \overline{2,k}, \\
\quad \left(to_{i-1,j_i,j}: po_3^{i-1,j}, pil_1^{i,j} \rightarrow pi_1^{i,j}, pol_3^{i-1,j}\right), i = \overline{2,k}, j = \overline{1,k}, \\
\left(to_{i-1,j-1_{,i},j}: po_7^{i-1,j-1}, pil_5^{i,j} \rightarrow pi_7^{i,j}, pol_5^{i-1,j-1}\right), i = \overline{2,k}, j = \overline{2,k}, \\
\left(to_{i,j-1,i-1,j}: po_6^{i,j-1}, pil_8^{i-1,j} \rightarrow pi_8^{i-1,j}, pol_6^{i,j-1}\right), i = \overline{2,k}, j = \overline{2,k}
\end{array}\right) \quad (12)$$

In Fig. 5, we show an example of an open 2×2 square lattice model with dedicated channels, represented by transitions connecting nodes, and cut-through switching nodes.

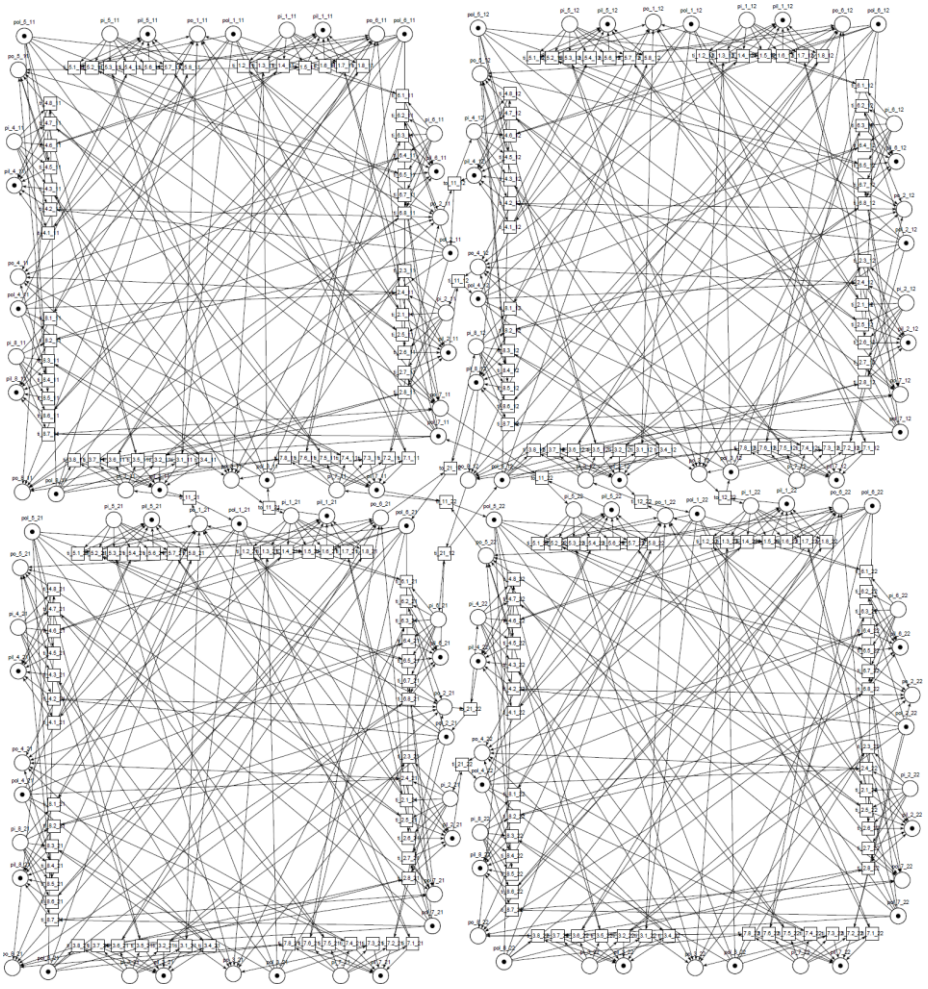


Fig. 5. Model of 2×2 open lattice with cut-through nodes with Moore's neighborhood and dedicated channels.

Table 2 specifies the model size of a square switching lattice of size k using the cut-through nodes with Moore's neighborhood and dedicated channels, measured in the number of places, transitions, and arcs.

Table 2. The open switching square lattice size.

Number of places	Number of transitions	Number of arcs
$32 \cdot k^2$	$56 \cdot k^2 + 2 \cdot 2 \cdot k \cdot (k\text{-}1) + 2 \cdot 2 \cdot (k\text{-}1)^2 =$ $64 \cdot k^2 - 12 \cdot k + 4$	$224 \cdot k^2 + 16 \cdot (2k\text{-}1) \cdot (k\text{-}1) =$ $256 \cdot k^2 - 48 \cdot k + 16$

We use magnitudes from Table 2 in subsection 4.3 for the comparisons with the number of places in the obtained *p*-invariants [30]. Model-checking techniques also enumerate the Petri net model elements within the square lattice.

4.3 Analysis of an open lattice with cut-through switching nodes

We use the parametric expressions (11), (12), represented in the direct form of the lattice model, similar to those studied in [10], for the analysis of model properties. For the calculation of open square lattice *p*-invariants [30], we compose, on (11) and (12), an infinite system of linear Diophantine algebraic equations (13); the system is represented in the parametric form with the parameter *k* having an infinite countable range.

$$\left(\left(\left(\left(ti_{u,v}^{i,j}\colon xpi_u^{i,j}+xpol_v^{i,j}-xpo_v^{i,j}-xpil_u^{i,j}=0\right),\begin{matrix}1\le v\le 8,\\ v\ne u\end{matrix}\right),1\le u\le 8\right),\begin{matrix}1\le i\le k,\\ 1\le j\le k\end{matrix}\right)$$

$$\left(\begin{array}{l}\left(ti_{i,j-1_i,j}\colon xpo_4^{i,j}+xpil_2^{i,j-1}-xpi_2^{i,j-1}-xpol_4^{i,j}=0\right), i=\overline{1,k}, j=\overline{2,k},\\ \left(ti_{i-1,j\_i\ \ ,j}\colon xpo_1^{i,j}+xpil_3^{i-1,j}-xpi_3^{i-1,j}-xpol_1^{i,j}=0\right), i=\overline{2,k}, j=\overline{1,k},\\ \left(ti_{i-1,j-1_{,i},j}\colon xpo_5^{i,j}+xpil_7^{i-1,j-1}-xpi_7^{i-1,j-1}-xpol_5^{i,j}=0\right), i=\overline{2,k}, j=\overline{2,k},\\ \left(ti_{i,j-1_{\_i}-1,j}\colon xpo_8^{i-1,j}+xpil_6^{i,j-1}-xpi_6^{i,j-1}-xpol_8^{i-1,j}\right), i=\overline{2,k}, j=\overline{2,k},\\ \left(to_{i,j-1_{\_i},j}\colon xpo_2^{i,j-1}+xpil_4^{i,j}-xpi_4^{i,j}-xpol_2^{i,j-1}=0\right), i=\overline{1,k}, j=\overline{2,k},\\ \left(to_{i-1,j\_i,j}\colon xpo_3^{i-1,j}+xpil_1^{i,j}-xpi_1^{i,j}-xpol_3^{i-1,j}=0\right), i=\overline{2,k}, j=\overline{1,k},\\ \left(to_{i-1,j-1_{,i},j}\colon xpo_7^{i-1,j-1}+xpil_5^{i,j}-xpi_7^{i,j}-xpol_5^{i-1,j-1}\right), i=\overline{2,k}, j=\overline{2,k},\\ \left(to_{i,j-1_i-1,j}\colon xpo_6^{i,j-1}+xpil_8^{i-1,j}-xpi_8^{i-1,j}-xpol_6^{i,j-1}\right), i=\overline{2,k}, j=\overline{2,k}\end{array}\right) \quad (13)$$

We calculate *p*-invariants of the IPN model (11), (12), and solve system (13) in the parametric form to find such properties as conservativeness and boundedness [30] for a given structure of any size. We obtain the parametric solution (14) of system (13) for the open square lattice with cut-through nodes, Moore's neighborhood, and dedicated channels given by (11) and (12).

$$\left(\begin{array}{c}\left(pi_u^{i,j},pil_u^{i,j}\right),1\le u\le 8,1\le i\le k,1\le j\le k,\\ \left(po_u^{i,j},pol_u^{i,j}\right),1\le u\le 8,1\le i\le k,1\le j\le k,\\ \left(\left(\left(pil_u^{i,j},pol_u^{i,j}\right),1\le u\le 8\right),1\le i\le k,1\le j\le k\right),\\ \left(\left(\left(pi_u^{i,j},po_u^{i,j}\right),1\le u\le 8\right),1\le i\le k,1\le j\le k\right)\end{array}\right) \quad (14)$$

We derived formulae to evaluate the number of obtained place invariants for specific values of *k*, which are gathered in Table 3.

Table 3. Number of basis invariants.

Invariant	Lines	Components
Invariant 1	$8\cdot k^2$	2
Invariant 2	$8\cdot k^2$	2

Invariant 3	1	$16 \cdot k^2$
Invariant 4	1	$16 \cdot k^2$

The total number of place invariants within a square lattice of size k with cut-through nodes, Moore's neighborhood, and dedicated channels is

$$N1=16 \cdot k^2 + 2. \qquad (15)$$

Each of the model places, totally $2 \cdot 8 \cdot k^2 + 2 \cdot 8 \cdot k^2 = 32 \cdot k^2$ places, enters the sum of the first and the second lines of (14) with a positive coefficient. Also, each of the model places, totally $16 \cdot k^2 + 16 \cdot k^2 = 32 \cdot k^2$ places according to Table 3, enters the sum of the third and fourth lines of (14) that proves *p*-invariance [30] of the model. As a collateral result, we evaluated in an alternative way the total number of the model places equal to $32 \cdot k^2$, the same as in Table 2.

4.4 Model composition of square lattice with combined cut-through and store-and-forward switching device and Moore's neighborhood

The model of an eight-port switching device with a combined switching mode of cut-through and store-and-forward procedures in the IPN form is shown in Fig. 6, ports are enumerated in the same way as for the cut-through model (Fig. 4). There are eight ports in the model; each port contains input and output channels. An internal buffer is described by eight buffer sections, one per port, respectively, and a buffer size limit. An example of a device model, shown in Fig. 6, contains five packets (tokens) in the port 5 buffer section, for a total of 55 packets in the switching node, and the available buffer size equal to 50. The following list of rules specifies the combined switching mode [41]:

– a packet is received by an input port of the switching node;
– as the next hop forwarding destination, a random output port, among seven output ports (excepts the current input port), is chosen;
– if the chosen output port is free, then the cut-through switching procedure forwards the packet to the output port directly without buffering;
– if the chosen output port is busy, then the store-and-forward switching procedure stores the packet in the internal buffer of node, namely in its section corresponding to the chosen output port;
– when an output port channel becomes free, the packet is forwarded to it from the corresponding section of the internal buffer in case the section is not empty.

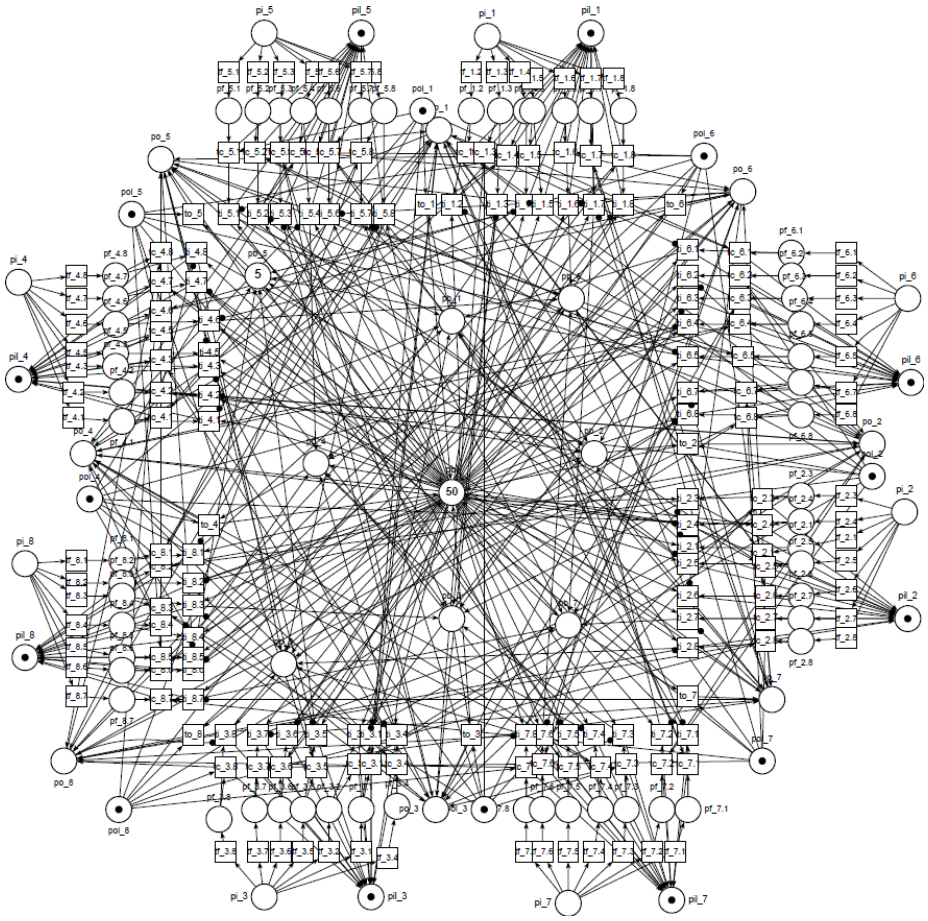


Fig. 6. A switching device model with Moore's neighborhood and combined cut-through and store-and-forward modes of switching.

We construct a direct parametric expression (16) of the device model shown in Fig. 6 to specify the device model in the PN form, which has four rows. Transitions of all input port tracts, having index v as a number of the next output port for the packet forwarding, are specified by the first three rows of (16). Transitions of all output port tracts, having index u as a number of the current port, are specified by the fourth row of (16). Remind that the letter "t" corresponds to a transition, the letter "p" corresponds to a place, and the letter "b" corresponds to a buffer.

$$\left(\left(\begin{pmatrix} tf_{u,v}: pi_u \to pf_{u,v} \\ tc_{u,v}: pf_{u,v}, pol_v \to po_v, pil_u \\ ti_{u,v}: pf_{u,v}, pbl, po_v \to po_v, pb_v, pil_u \end{pmatrix}, 1 \le v \le 8, v \ne u\right), 1 \le u \le 8\right) \quad (16)$$

We use parametric expression (16), which represents the device model shown in Fig. 6 to compose a square communication lattice model in the form of IPN. Note that we specify regular structures only, for instance represented by a lattice.

We add letters, which correspond to the rules of switching, to the names of places and transitions of the device model. The letter "*f*" stands for packet forwarding:

$tf_{u,v}$ – a decision on the packet forwarding from the current port *u* to the output port *v*;

$pf_{u,v}$ – the indicator of forwarding.

The letter "*c*" denoted the cut-through switching mode:

$tc_{u,v}$ – a direct packet forwarding from the input port to the output port;

$ti_{u,v}$ – store a packet within the corresponding section of the internal buffer.

The letter "i" denotes an input tract of the device port; the letter "o" specifies an output tract of the device port. For example, a transition to_u outputs a packet from the section *u* of the internal buffer to the corresponding port.

The input port buffer and the limitation of input port buffer size (a unit) are specified by places pi_u and pil_u, respectively, while the output port buffer and the limitation of output port buffer size (a unit) are specified by places po_u (po_v) and pol_u (pol_v), respectively. The internal buffer section of the port with index *v* is denoted as pb_v and the device buffer limitation is specified by the place pbl marking. The numbers of tokens in all places pb_v and place pbl are complementary, with their invariant equal to the internal buffer size.

Transition $ti_{u,v}$ has a read arc directed from the destination port buffer po_v; here the read arc is an abbreviation of a loop created by a pair of arcs having counter directions, connecting place po_v and transition $ti_{u,v}$. We represent the lower priority of buffering compared to cut-through forwarding via the read arc. Only in the case when the destination port is busy, the buffering is implemented. The choice between transmitting the arrived packet or a packet from the buffer is implemented in a nondeterministic way when there are packets within a certain section of the internal buffer and an input packet forwarded to the same port. It seems to be the best simple solution that does not lead to sophisticated models. The solution is also justified by real-life procedures in case we do not model priority classes of packets.

We calculated the number of places, transitions, and arcs within the combined switching device model and represented them with Table 4.

Table 4. Number of places, transitions, and arcs within the combined switching device model.

Number of places	Number of transitions	Number of arcs
(7+4)·8+9=	(7+7+7+1)·8=	(7+7+2·7+3·7+4·7+4)·8=
97	176	648

Evaluations from Table 4 are applied in sequel for computing the corresponding numbers for the square lattice model.

4.5 Specification of an open lattice with combined switching mode

In Fig. 7, a visual representation of an open lattice model of size 3×3 with a combined cut-through and store-and-forward switching device is shown.

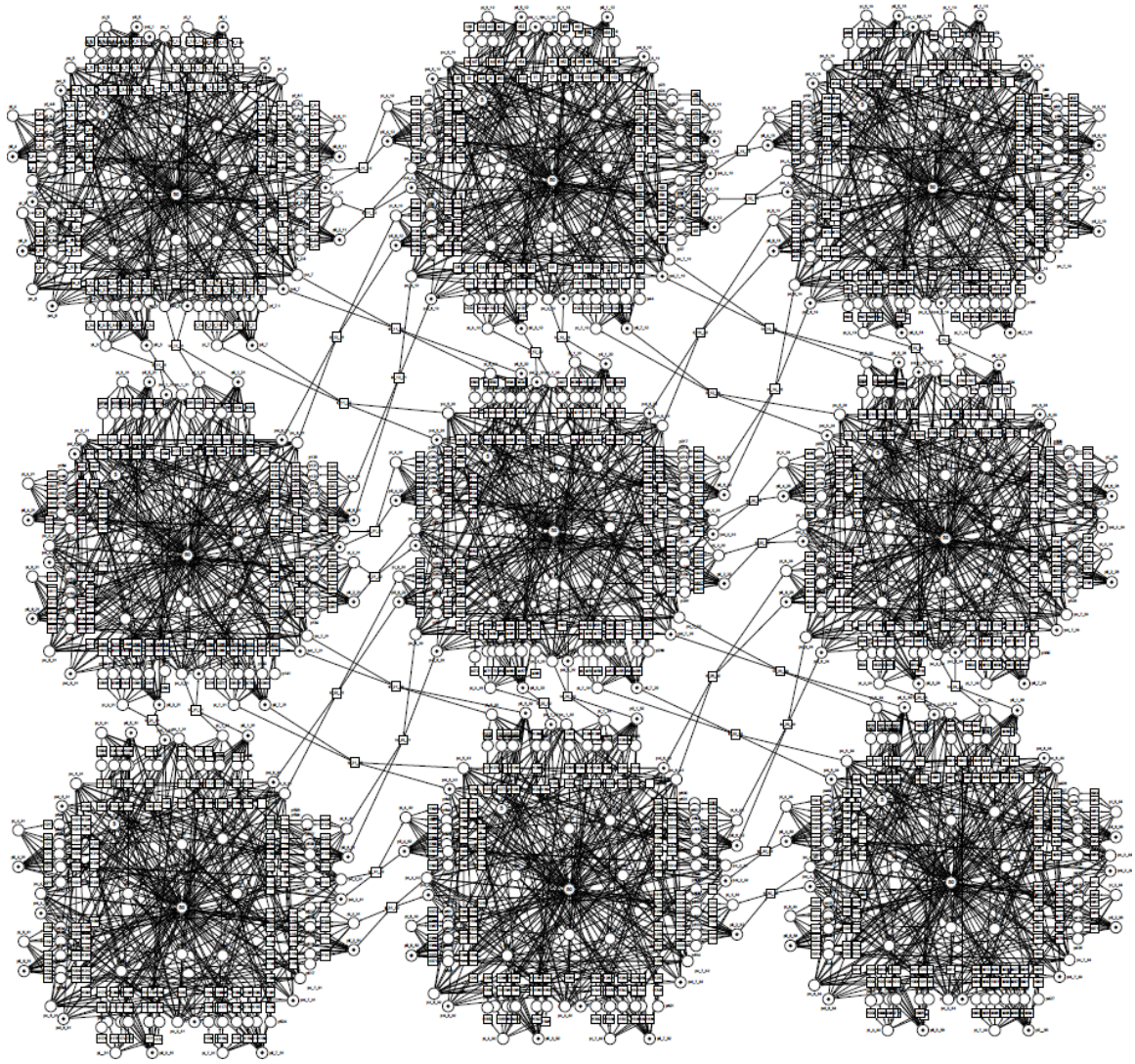


Fig. 7. A model of 3×3 open square lattice with combined switching node.

We compose an open square communication lattice of size k based on the direct parametric expression (16) of the device model shown in Fig. 6. Indexes *(i, j)* have been added to the parametric description of the eight-port communication device model (16) to obtain the lattice specification (17), where index i is the row number and index j is the column number of the device within the lattice.

$$\left(\left(\left(\begin{array}{l} tf_{u,v}^{i,j}: pi_u^{i,j} \to pf_{u,v}^{i,j} \\ tc_{u,v}^{i,j}: pf_{u,v}^{i,j}, pol_v^{i,j} \to po_v^{i,j}, pil_u^{i,j} \\ ti_{u,v}^{i,j}: pf_{u,v}^{i,j}, pbl^{i,j}, po_v^{i,j} \to po_v^{i,j}, pb_v^{i,j}, pil_u^{i,j} \\ to_u^{i,j}: pb_u^{i,j}, pol_u^{i,j} \to po_u^{i,j}, pbl^{i,j} \end{array}\right), \begin{array}{c} 1 \le v \le 8, \\ v \ne u \end{array}\right), 1 \le u \le 8\right), \begin{array}{l} 1 \le i \le k, \\ 1 \le j \le k \end{array} \quad (17)$$

All nodes of the $k \times k$ lattice are described by the specification (17), while all connections between nodes of the lattice are described by the specification (12) similar to the lattice with

the cut-through communication node. Based on the composition rules, the specification of an open lattice model with a combined switching node is represented by expressions (18),(19); here we repeated the formulae for the convenience of subsequent composition of a linear equation system.

$$\left(\left(\left(\begin{array}{l} tf_{u,v}^{i,j}: pi_u^{i,j} \to pf_{u,v}^{i,j} \\ tc_{u,v}^{i,j}: pf_{u,v}^{i,j}, pol_v^{i,j} \to po_v^{i,j}, pil_u^{i,j} \\ ti_{u,v}^{i,j}: pf_{u,v}^{i,j}, pbl^{i,j}, po_v^{i,j} \to po_v^{i,j}, pb_v^{i,j}, pil_u^{i,j} \end{array}\right), \begin{array}{c} 1 \le v \le 8, \\ v \ne u \end{array}\right), 1 \le u \le 8\right), \begin{array}{l} 1 \le i \le k, \\ 1 \le j \le k \end{array}$$

$$to_u^{i,j}: pb_u^{i,j}, pol_u^{i,j} \to po_u^{i,j}, pbl^{i,j} \tag{18}$$

$$\left(\begin{array}{l} \left(ti_{i,j-1_i,j}: po_4^{i,j}, pil_2^{i,j-1} \to pi_2^{i,j-1}, pol_4^{i,j}\right), i = \overline{1,k}, j = \overline{2,k}, \\ \quad \left(ti_{i-1,j\_i\ \ ,j}: po_1^{i,j}, pil_3^{i-1,j} \to pi_3^{i-1,j}, pol_1^{i,j}\right), i = \overline{2,k}, j = \overline{1,k}, \\ \left(ti_{i-1,j-1_{,i},j}: po_5^{i,j}, pil_7^{i-1,j-1} \to pi_7^{i-1,j-1}, pol_5^{i,j}\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(ti_{i,j-1\_i-1,j}: po_8^{i-1,j}, pil_6^{i,j-1} \to pi_6^{i,j-1}, pol_8^{i-1,j}\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(to_{i,j-1\_i,j}: po_2^{i,j-1}, pil_4^{i,j} \to pi_4^{i,j}, pol_2^{i,j-1}\right), i = \overline{1,k}, j = \overline{2,k}, \\ \quad \left(to_{i-1,j\_i,j}: po_3^{i-1,j}, pil_1^{i,j} \to pi_1^{i,j}, pol_3^{i-1,j}\right), i = \overline{2,k}, j = \overline{1,k}, \\ \left(to_{i-1,j-1_{,i},j}: po_7^{i-1,j-1}, pil_5^{i,j} \to pi_7^{i,j}, pol_5^{i-1,j-1}\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(to_{i,j-1_i-1,j}: po_6^{i,j-1}, pil_8^{i-1,j} \to pi_8^{i-1,j}, pol_6^{i,j-1}\right), i = \overline{2,k}, j = \overline{2,k} \end{array}\right) \tag{19}$$

Formulae to compute the number of places, transitions, and arcs within the open square lattice model with Moore's neighborhood and combined switching node are gathered in Table 5.

Table 5. Number of places, transitions, and arcs within the open lattice with a combined node.

Number of places	Number of transitions	Quantity of arcs
	$176 \cdot k^2 + 4 \cdot k \cdot (k-1) + 4 \cdot (k-1)^2 =$	$648 \cdot k^2 + 16 \cdot (2k-1) \cdot (k-1) =$
$97 \cdot k^2$	$184 \cdot k^2 - 12 \cdot k + 4$	$680 \cdot k^2 - 48 \cdot k + 16$

The evaluated numbers of vertices and arcs have been indirectly confirmed by the modeling system Tina for a series of finite sizes represented by the values of parameter *k*.

4.6 Resolving infinite linear system for place invariants in parametric form

For the calculation of *p*-invariants of the open lattice (18), (19), we composed an infinite system of linear Diophantine algebraic equations (20), (21) in parametric form.

$$\left(\left(\left(\left(\begin{pmatrix} tf_{u,v}^{i,j}: pi_u^{i,j} - pf_{u,v}^{i,j} = 0 \\ tc_{u,v}^{i,j}: pf_{u,v}^{i,j} + pol_v^{i,j} - po_v^{i,j} - pil_u^{i,j} = 0 \\ ti_{u,v}^{i,j}: pf_{u,v}^{i,j} + pbl^{i,j} + po_v^{i,j} - po_v^{i,j} - pb_v^{i,j} - pil_u^{i,j} = 0 \end{pmatrix}, \begin{matrix} 1 \le v \le 8, \\ v \ne u \end{matrix}\right), 1 \le u \le 8\right), \begin{matrix} 1 \le i \le k, \\ 1 \le j \le k \end{matrix}\right)$$

$$to_u^{i,j}: pb_u^{i,j} + pol_u^{i,j} - po_u^{i,j} - pbl^{i,j} = 0 \quad (20)$$

$$\left(\left(\begin{matrix} \left(ti_{i,j-1_i,j}: xpo_4^{i,j} + xpil_2^{i,j-1} - xpi_2^{i,j-1} - xpol_4^{i,j} = 0\right), i = \overline{1,k}, j = \overline{2,k}, \\ \left(ti_{i-1,j\_i\ \ ,j}: xpo_1^{i,j} + xpil_3^{i-1,j} - xpi_3^{i-1,j} - xpol_1^{i,j} = 0\right), i = \overline{2,k}, j = \overline{1,k}, \\ \left(ti_{i-1,j-1_{,i},j}: xpo_5^{i,j} + xpil_7^{i-1,j-1} - xpi_7^{i-1,j-1} - xpol_5^{i,j} = 0\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(ti_{i,j-1_{-i}-1,j}: xpo_8^{i-1,j} + xpil_6^{i,j-1} - xpi_6^{i,j-1} - xpol_8^{i-1,j}\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(to_{i,j-1_{-i},j}: xpo_2^{i,j-1} + xpil_4^{i,j} - xpi_4^{i,j} - xpol_2^{i,j-1} = 0\right), i = \overline{1,k}, j = \overline{2,k}, \\ \left(to_{i-1,j_{-i},j}: xpo_3^{i-1,j} + xpil_1^{i,j} - xpi_1^{i,j} - xpol_3^{i-1,j} = 0\right), i = \overline{2,k}, j = \overline{1,k}, \\ \left(to_{i-1,j-1_{,i},j}: xpo_7^{i-1,j-1} + xpil_5^{i,j} - xpi_7^{i,j} - xpol_5^{i-1,j-1}\right), i = \overline{2,k}, j = \overline{2,k}, \\ \left(to_{i,j-1_i-1,j}: xpo_6^{i,j-1} + xpil_8^{i-1,j} - xpi_8^{i-1,j} - xpol_6^{i,j-1}\right), i = \overline{2,k}, j = \overline{2,k} \end{matrix}\right)\right) \quad (21)$$

The following parametric solution (22) of system (20), (21) was obtained for the open communication square lattice of size k (18) having a combined switching node:

$$\left(\begin{matrix} \left(po_u^{i,j}, pol_u^{i,j}\right), 1 \le u \le 8, 1 \le i \le k, 1 \le j \le k \\ \left(\left(pb_u^{i,j}, 1 \le u \le 8\right), pbl^{i,j}\right), 1 \le i \le k, 1 \le j \le k \\ \left(\left(pf_{u,v}^{i,j}, 1 \le v \le 8, v \ne u\right), pi_u^{i,j}, pil_u^{i,j}\right), 1 \le u \le 8, 1 \le i \le k, 1 \le j \le k \\ \left(\left(pbl^{i,j}, \left(\left(pil_u^{i,j}, pol_u^{i,j}\right), 1 \le u \le 8\right), 1 \le i \le k, 1 \le j \le k\right)\right. \\ \left(\left(pb_u^{i,j}, \left(pf_{u,v}^{i,j}, 1 \le v \le 8, v \ne u\right), pi_u^{i,j}, po_u^{i,j}\right), 1 \le u \le 8, 1 \le i \le k, 1 \le j \le k\right) \end{matrix}\right) \quad (22)$$

Table 6 contains the calculated numbers of basis invariants of various types specified with formulae.

Table 6. Number of basis invariants in parametric sparse matrix (22).

Invariant	Number of lines	Components
Invariant 1	$8 \cdot k^2$	2
Invariant 2	k^2	9
Invariant 3	$8 \cdot k^2$	9
Invariant 4	1	$17 \cdot k^2$
Invariant 5	1	$80 \cdot k^2$

We conclude that the total number of *p*-invariants in a lattice with a combined switching node of size *k* is

$$N2=17\cdot k^2 + 2. \qquad (23)$$

All the places, $2\cdot 8\cdot k^2 + 9\cdot k^2 + 9\cdot 8\cdot k^2 = 97\cdot k^2$, are contained as natural components of the sum of the first, second, and third lines of (22). Also, all the places, $17\cdot k^2 + 80\cdot k^2 = 97\cdot k^2$, are contained as natural components of the sum of the fifth and fourth lines of (22) according to Table 6. Consequently, the number of places in the open square lattice model with a combined node of size *k* is $97\cdot k^2$ according to Tables 5 and 6.

4.7 Proof of the model properties

In this section, we prove the conservativeness and boundedness [30] of the square lattice model with a cut-through switching node and Moore's neighborhood represented as an infinite Petri net. We prove that the Petri net is *p*-invariant and has the properties of conservativeness and boundedness for an arbitrary integer *k* when each line of the sparse matrix (14) is a solution of system (13).

Lemma 1. Each line of the sparse matrix (14) is a solution of the sparse linear system of Diophantine equations (13).

Proof. We substitute each parametric line of the parametric solution (14) into the each parametric equality of the equation system (13). Various values of indices are selected when substituting; as a result, the correct statement is obtained. For instance, when we substitute the second line of the parametric solution (14)

$$\left(po_u^{i,j}, pol_u^{i,j}\right), 1 \le u \le 8, 1 \le i \le k, 1 \le j \le k,$$

into the first equation of the system (13)

$$\left(\left(\left(ti_{u,v}^{i,j}: xpi_u^{i,j} + xpol_v^{i,j} - xpo_v^{i,j} - xpil_u^{i,j} = 0\right), \begin{matrix} 1 \le v \le 8, \\ v \ne u \end{matrix}\right), 1 \le u \le 8\right), \begin{matrix} 1 \le i \le k, \\ 1 \le j \le k \end{matrix}$$

we obtain the following equalities:

for $i' \ne i$ or $j' \ne j$: $0 + 0 - 0 - 0 = 0$, we obtain $0 = 0$;

for $i' = i$ or $j' = j$: $0 + 1 - 1 - 0 = 0$, we obtain $0 = 0$.

We check all other combinations in the same way.

□

Theorem 1. The lattice model (11), (12) is a *p*-invariant Petri net for an arbitrary size given by natural number *k*.

Proof. Let us calculate the sum of the first and the second lines of (14); it contains all the places once, $2\cdot 8\cdot k^2 + 2\cdot 8\cdot k^2 = 32\cdot k^2$; and the sum of the third and fourth lines of (15) contains all the places once, $16\cdot k^2 + 16\cdot k^2 = 32\cdot k^2$, that coincides with Table 3. The number of places in the open square lattice model of size *k* is $32\cdot k^2$ which coincides with Table 2. These lines are solutions of system (13) according to *Lemma 1*.

□

Thus, we calculated place invariants [30] of infinite Petri nets [14,15] and analyzed properties [42] of an ideal communication protocol model based on the parametric expressions (11), (12), which specify the communication square lattice of any size *k* with

Moore's neighborhoods [21,23] and cut-through switching node. In a similar way, we can prove properties of the model (18), (19) with a combined switching node.

5 Specialized Computing Lattice Model Composition and Implementation on FPGA

For the embedded application domain, we need a device numerically resolving a PDE, which specifies the control task model, in a few microseconds; for special applications, in a few nanoseconds, that is infeasible for microcontrollers. Here we implement the mass-parallel specifications, developed in Sections 3, 4 based on the iteration process of high accuracy given by formulae (8)-(9) in Section 2, in hardware, preserving the model parallelism to achieve the utmost performance required for control of fast real-time processes in manifold applications of physics, chemistry, and engineering.

5.1 Specialized computing-communication lattice model

For embedded applications [6], we develop a specialized integrated model of computation and communication in the form of a lattice with the number of nodes corresponding to the required accuracy of the finite difference method. The iteration technique allows us to simplify the node function considerably, and the application of the synchronous communication scheme reduces the communication subsystem complexity. In this section, we prototype the mass-parallel hardware in FPGA [11], generating Verilog [12] code on the lattice abstract specification. For special applications, dedicated integrated circuits can be designed.

The present subsection focuses on composing models for subsequent hardware implementation, while the next subsection presents techniques for Verilog code generation. We start by specializing the model of Section 3, shown in Fig. 3, to adjust it for a direct synchronous connection of nodes employing the minimal number of the target device time cycles for the iteration implementation. In Fig. 8, we represent the model, drawn in Fig. 3, in a reduced form, transmitting the node value after its recalculation directly to the node neighbors, according to Moore's neighborhood [21,23], for the next iteration.

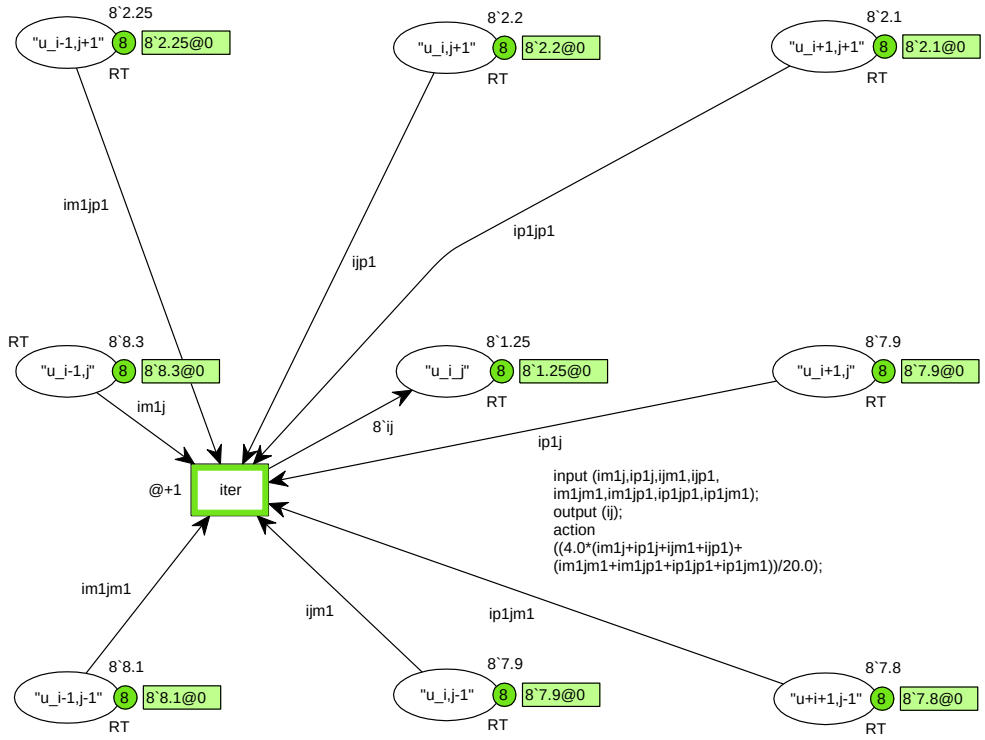


Fig. 8. CPN model of a node for further composition of lattice via merging places of neighboring nodes.

We use the node model shown in Fig. 8 for the lattice composition via collecting models of indexed nodes and merging places having the same name. Formal specification of the lattice composition via merging (uniting) contact places of components is studied in [18,43]. In Fig. 9, we provide an example of the 5×5 lattice, supposing constant border conditions given by the initial marking. It means that the border conditions do not change during the iteration process, though they can be different as defined by the corresponding border condition functions for different border nodes. Since the rectangle perimeter represents the constant border conditions, we have an internal grid of 3×3 nodes containing transitions, which implement the iteration process. For brevity and vividness of graphical representation, we omit the transition actions that are the same (as shown in Fig. 8) and can be specified by the corresponding function similar to IP network address extraction [36].

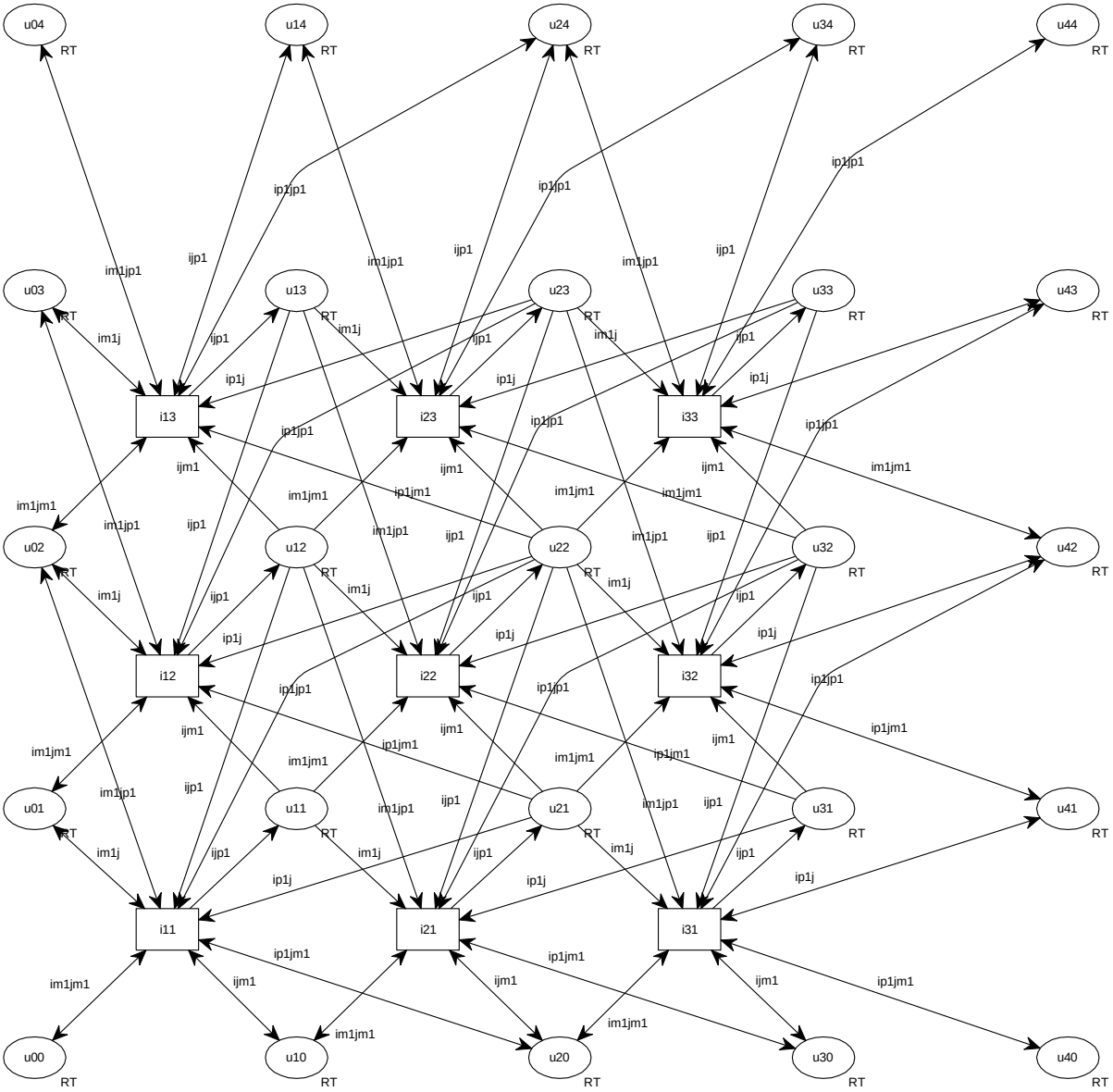


Fig. 9. A 5x5 lattice CPN model with 3 × 3 internal nodes; the model specifies the computation and synchronous communication to recalculate all the nodes of a finite difference approximation in one time cycle.

For real-life numerical solving of PDEs, big lattices are employed. To specify such lattices, we offer two basic approaches. The first approach supposes developing special software for generating big models similar to the previously developed generators for various types of communication lattices [15]. A generator accepts, as an input parameter, the grid size, possibly, the number of dimensions for multidimensional lattices, neighborhood specification, and the iteration formula together with the border conditions formulae. The generator output represents the lattice specification according to the chosen modeling system, either in a graphical form or an abstract form without graphical binding, more appropriate for multidimensional lattices that are difficult to visualize. The result of generator work

represents the technological specifications for hardware implementation of a specialized computer for solving a given PDE using a given finite difference scheme via the iteration process.

The second approach bears a more abstract and descriptive character suitable for both direct implementation on computing clusters and synthesizing dedicated hardware. We call it a reenterable model [18], where each component is represented in a single copy, tokens are supplied with a topological tag, and movements, with respect to real-life topology, are modeled by the process of tag switching. The reenterable model shown in Fig. 10 is the same for a lattice of any size; only the marking changes based on a given size. For multidimensional lattices, the number of dimensions can also be specified with a parameter. Note that this kind of model looks more suitable for the butterfly topology of HPC interconnects provided by Infiniband switches such as Slingshot [27] because they do not depend directly on a specific mesh of physical connections.

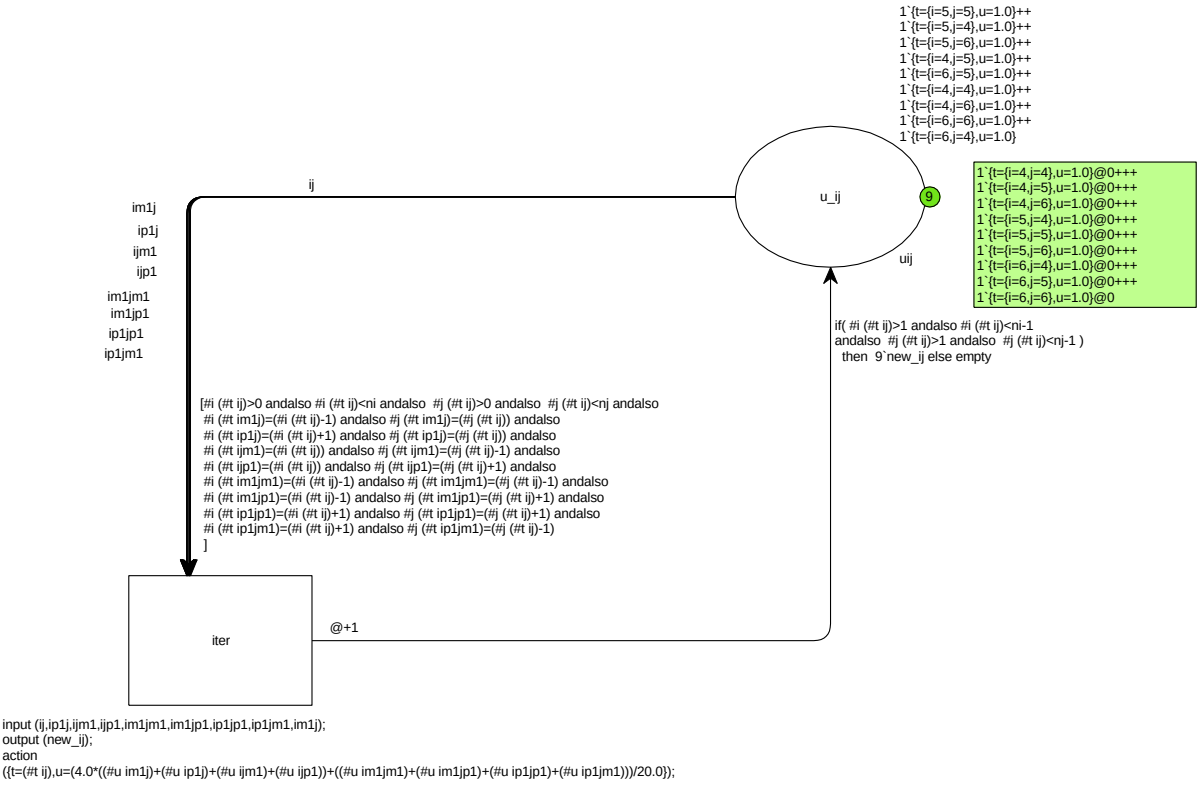


Fig. 10. Reenterable model of a lattice of any size; values are supplied with the lattice coordinate tags, transition *iter* represents recalculation of all the nodes values.

A reenterable model represents a rather high level of abstraction, especially when multidimensional lattices are specified [15,20] convenient for the description purposes, though it requires optimization for its efficient implementation. The optimization facilities are not provided within CPN Tools [17]. In Fig. 10, the function values at the current iteration are stored within a single place *"u_ij"*, recalculation of all the node values is implemented via a single transition *iter*. The following declarations are employed:

```
val ni=10; val nj=10;
```

```
colset ri=int with 0..ni; colset rj=int with 0..nj;
colset tag=record i:ri * j:rj;
colset uij=record t:tag * u:REAL timed;
var ij,im1j,ip1j,ijm1,ijp1,im1jm1,im1jp1,ip1jp1,ip1jm1,new_ij: uij;
```

Predefined constants (values) *ni* and *nj* specify the lattice size, data types *ri* and *rj* specify the coordinate ranges that can be employed in generators of initial marking using *ri.all()* and *rj.all()* constructs. The data type *tag* specifies the coordinates of a node while *uij* defines the node value.

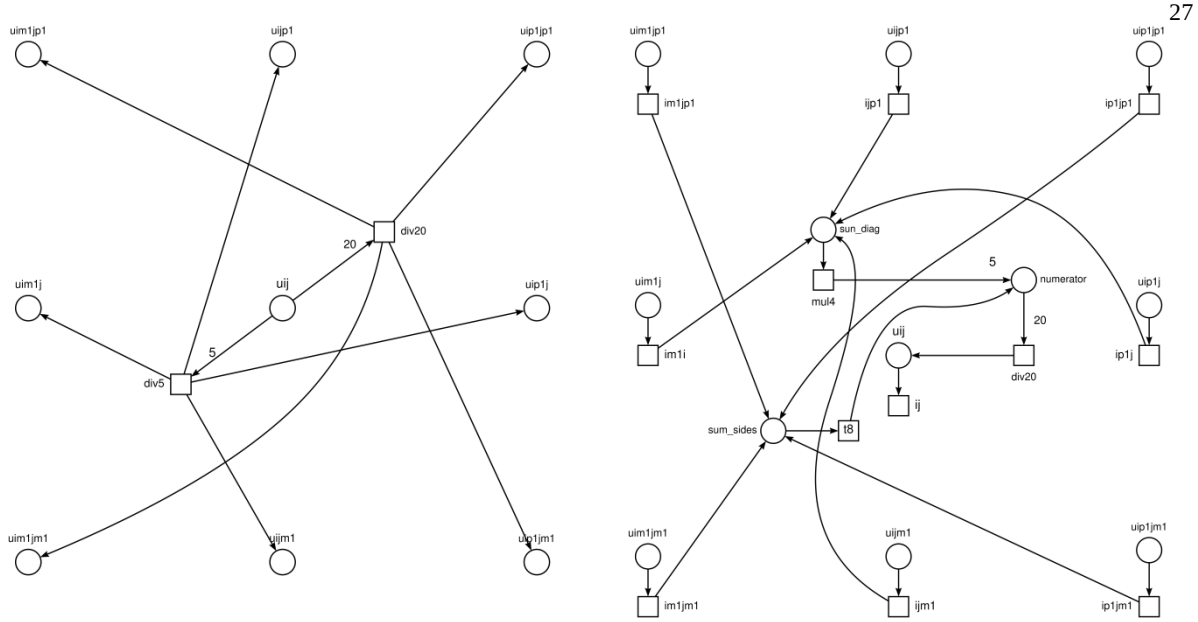
In Fig. 10, the complex guard function of transition *iter* selects values of 9 neighboring nodes via the set of variables *ij,ip1j,ijm1,ijp1,im1jm1,im1jp1,ip1jp1,ip1jm1,im1j* satisfying the specified condition on their indices according to Moore's neighborhood [21]; it also selects as *ij* an internal node only. Based on these values, the action of transition *iter* calculates a new value of the current node according to the formula (8) and assigns it to the variable *new_ij* to replace a token extracted by variable *ij*. The inscription of the outgoing arc excludes the border condition recalculation.

5.2 Synthesis of Specialized Lattices on FPGA

For the most significant tasks of real-time embedded control, we develop a specialized model of computations that is implemented in a semi-hardware form using FPGA [11,44], where generated Verilog [12] code defines the specialized hardware. We follow a traditional for embedded applications [6] way of approximating real numbers as fixed-point numbers or, in even more simplified form, as integer numbers, supposing an initial scaling of numbers with respect to the working range and required accuracy of the function approximation. For this purpose, a UNIT_VALUE is selected; for instance, if a unit equals 0.001, then all the values are expressed with respect to the chosen unit. For using unsigned numbers, the corresponding shift of values is applied; for instance, having a range from -10 to 10 and the above unit, we implement a shift by 10, having a working range of values from 0 to 20000.

Formal issues of convergence for nonnegative integer approximation require further investigation, and with such an approximation, we can directly apply conventional Petri and Sleptsov nets, which are easy for hardware implementation. Note that there is an intrinsic connection of cellular automata (CA) [22] with the iteration technique, especially when approximated with nonnegative integer numbers. Indeed, the node value, together with a given neighborhood and iteration formula, represents the CA rule [21]. Thus, we can apply CA theory [21] to study the convergence issues [2]. An unchanged CA configuration directly represents convergence, though some cycles of configurations with low differences in the node values are possible to achieve as well.

In Fig. 11, we present two different approaches to modeling the iteration processes by SNs; the first approach (Fig. 11a) is fast and rough, though it yields simpler constructs; the second approach (Fig. 11b) is more precise, though it runs three times slower and contains more graphical elements. The node model represents a clan (functional subnet) assuming applicability of the clan composition technique [36,45] for the model properties analysis.



(a) one time cycle per iteration; (b) three time cycles per iteration.

Fig. 11. Node model for integer number approximation lattice for embedded systems design.

Both nodes of Fig. 11 implement the iteration formula (8), the node shown in Fig. 11b implements it directly, while the node shown in Fig. 11a implements it with transformations (24) as a sum of 8 addendums in the third line; an intermediate variant is possible corresponding to the second line and requiring two time cycles.

$$u_{i,j}^{(s+1)} = \frac{4\left(u_{i,j-1}^{(s)} + u_{i-1,j}^{(s)} + u_{i,j+1}^{(s)} + u_{i+1,j}^{(s)}\right) + u_{i-1,j-1}^{(s)} + u_{i+1,j+1}^{(s)} + u_{i-1,j-1}^{(s)} + u_{i+1,j-1}^{(s)}}{20} =$$

$$\frac{u_{i,j-1}^{(s)}+u_{i-1,j}^{(s)}+u_{i,j+1}^{(s)}+u_{i+1,j}^{(s)}}{5} + \frac{u_{i-1,j-1}^{(s)}+u_{i+1,j+1}^{(s)}+u_{i-1,j-1}^{(s)}+u_{i+1,j-1}^{(s)}}{20} = \tag{24}$$

$$\frac{u_{i,j-1}^{(s)}}{5} + \frac{u_{i-1,j}^{(s)}}{5} + \frac{u_{i,j+1}^{(s)}}{5} + \frac{u_{i+1,j}^{(s)}}{5} + \frac{u_{i-1,j-1}^{(s)}}{20} + \frac{u_{i+1,j+1}^{(s)}}{20} + \frac{u_{i-1,j-1}^{(s)}}{20} + \frac{u_{i+1,j-1}^{(s)}}{20}.$$

For the node shown in Fig. 11a, a 5×5 lattice is composed in Fig. 12. It runs one iteration in one time cycle for the entire matrix of values for the synchronous implementation of SN (Sleptsov-Salwicki rule [32]). The lattice design is based on the transformations of formula (8) represented with (24).

Each of eight addendums is implemented by a separate transition of a neighboring node in Fig. 11a; separating division operations affects the accuracy of computations, though it leads to a one-time-cycle implementation with enhanced performance. Actually, a transition divides the node value by 5 or 20 and adds the obtained values to the neighboring nodes; the division is implemented using the corresponding weight of the transition incoming arc, while the addition is implemented implicitly with a few incoming arcs of the corresponding place. As it happens, the boundary nodes are not recalculated, while each internal node place has 8 incoming arcs bringing the corresponding addendums of the final expression in the transformations (24).

If this fastest scheme is not suitable from the error of the integer division point of view, a three-time cycle implementation strictly following formula (8) will be the best choice for

nonnegative integer approximation. The corresponding node model to compose a lattice is represented in Fig. 11b. For verification of SN programs, we apply system Tina [46] which recently implements SNs and represents an IDE [34] for our novel SNC paradigm of computations [13,47].

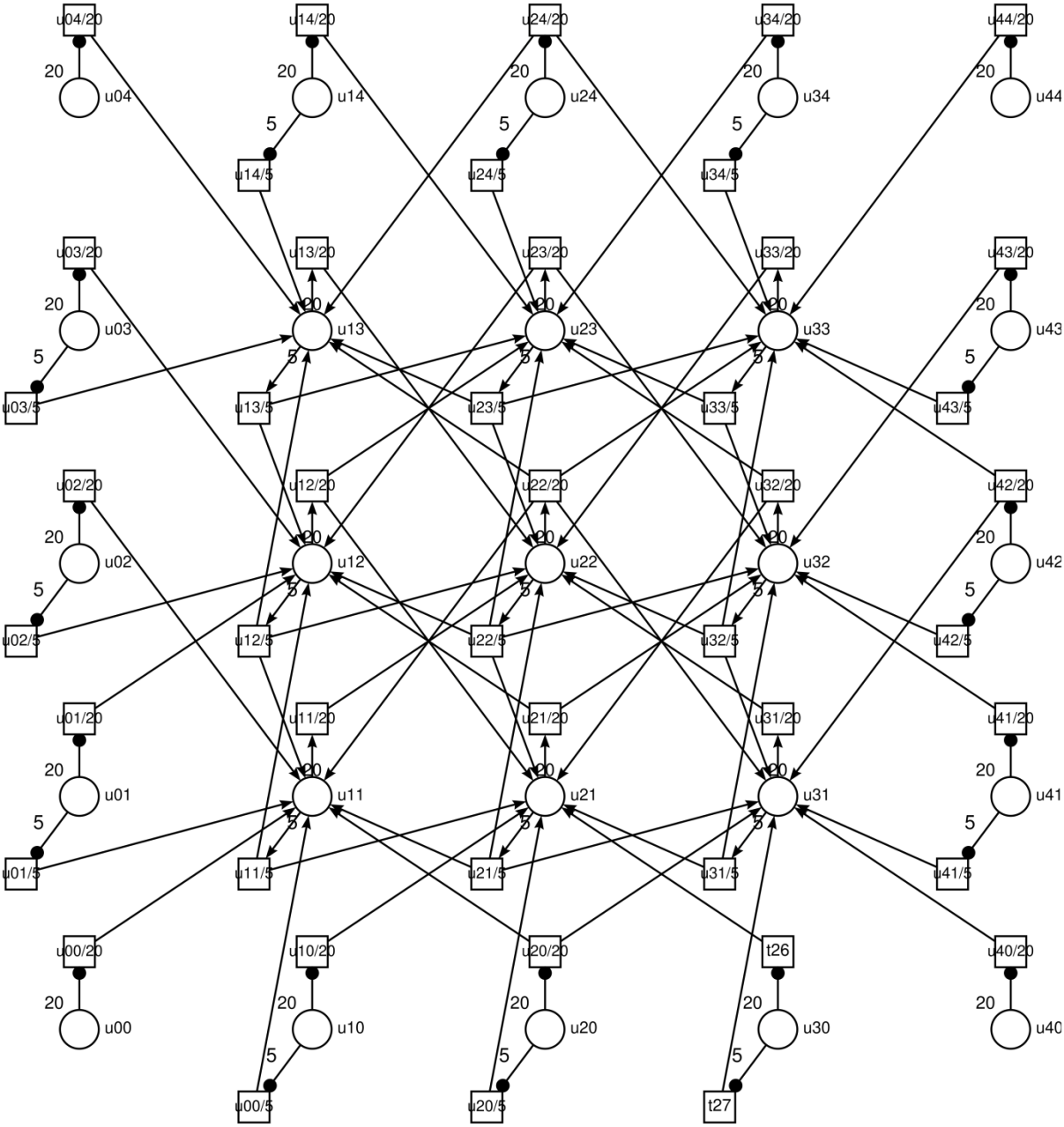


Fig. 12. An integer number approximation lattice solving PDE for embedded systems design with a node shown in Fig. 11a.

The composed SN model represents the lattice specification for further implementation of the approach [48] to compile an SN into Verilog code [11] for its subsequent implementation on FPGA [12]. We employ either direct mapping of the lattice into explicit Verilog statements or use loops as a template with respect to the lattice size parameter *N*. The

corresponding code snippet is shown in Fig. 13. On the growing edge of the clock signal, we recalculate the entire lattice for *K* iterations and then indicate with the built-in LED array the result for an internal node. We run a single iteration during a time cycle; the subsequent Verilog code fine-tuning supposes adjusting the iteration time with the device clock with the purpose to run the maximal possible number of iterations during a single time cycle. Since the conventional division of integer numbers just omits the fractional part, we need to provide a correction for better rounding. Because of this reason, we compare the division reminder *num%20* with a given constant *ROUND_EDGE* to add an extra unit. As a result of fine-tuning, we can recommend using the constant *ROUND_EDGE* values from 10 to 12.

```
always @(posedge sys_clk) begin
     if (counter < `K ) begin
        for(i = 1; i < `N-1; i = i+1) begin // iterate
          for(j = 1; j < `N-1; j = j+1) begin
            cross=u[i+1][j]+u[i-1][j]+u[i][j+1]+u[i][j-1];
            diag=u[i+1][j+1]+u[i+1][j-1]+u[i-1][j+1]+u[i-1][j-1];
            num=4*cross+diag;
            u1[i][j] = num/20+((num%ROUND_EDGE)<11)?0:1;
          end
        end
        for(i = 1; i < `N-1; i = i+1) begin // u=u1;
          for(j = 1; j < `N-1; j = j+1) begin
            u[i][j] = u1[i][j];
          end
        end

     end
     else
       led = ~ u[2][2][5:0];
     end
```

Fig. 13. Listing of Verilog code snippet of the iteration (8) hardware implementation.

An adjusted constant number of iterations, for a certain range of considered magnitudes, represents a good compromise with respect to the limited resources of embedded applications. A layout of a node, obtained as a result of Verilog code processing, is shown in Fig. 14; here we use neighboring node values according to the names specified in Figs. 8 – 11; it just describes the data flow of formula (8) with a 16-bit unsigned integer approximation.

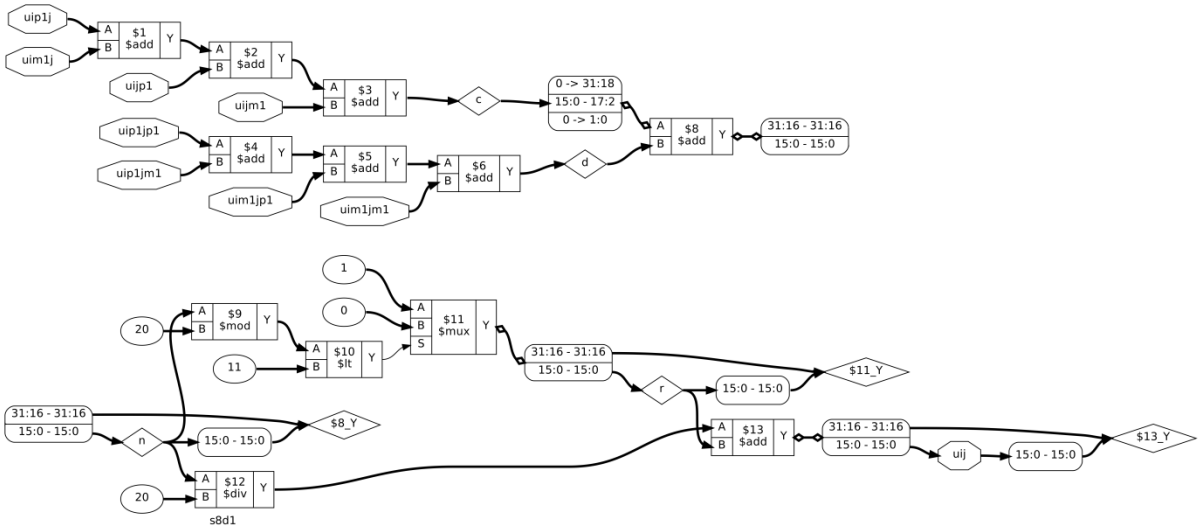


Fig. 14. Layout of a node obtained on Verilog code (automatic).

We tested the node (Fig. 14) and composed a lattice for $N=5$ that recalculates the value of its internal part represented by a 3×3 lattice using the perimeter nodes as constant boundary conditions. The lattice layout is shown in Fig. 15 and represents the connection of 9 nodes. Here we do not rearrange the automatically obtained layout, though when optimizing it for production, the lattice structure will be imposed, providing the minimal wire (connections) length and the minimal number of intersections to trace on additional layers.

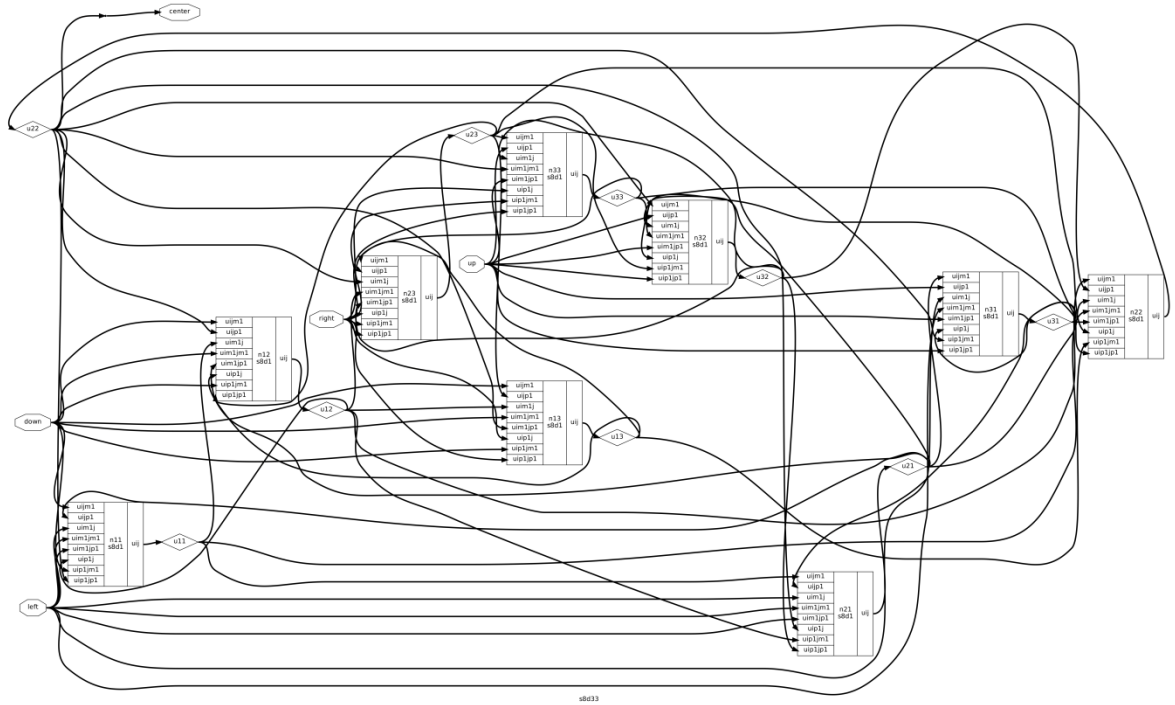
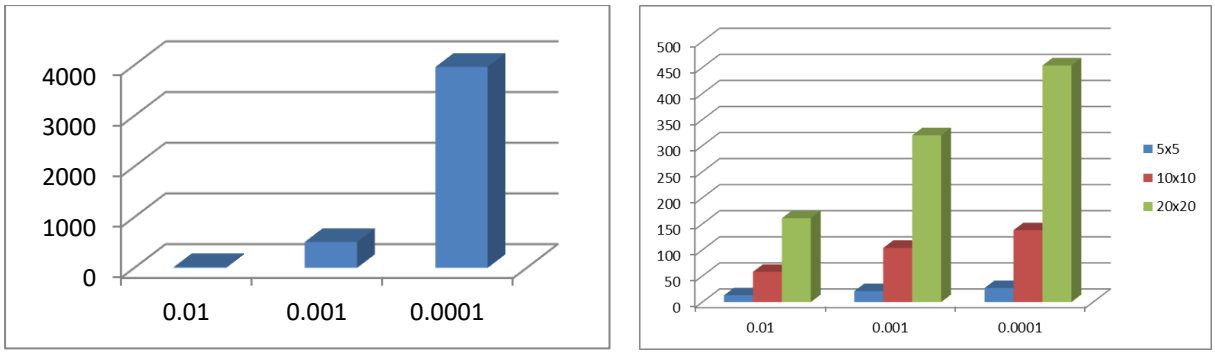


Fig. 15. Layout of 5x5 lattice with constant values on borders (automatic).

5.3 Some benchmarks for FPGA implementation

For preliminary benchmarks, we were using the Tang Nano 9k FPGA operated by Gowin FPGA Designer [48]. Dependence of the solution time on the lattice size for required precision is shown in Fig. 16. For the 50 MHz FPGA clock, we run an iteration in about 20 *ns*; thus, we calculate about *K=50* iterations in 1 *us* that provides a rather good approximation of the optimal control specified by PDE (a system of PDEs). Benchmarks obtained for the specified FPGA completely correspond to simplified evaluations by multiplying the number of iterations by the device's basic time cycle. Modern advanced FPGAs work on frequencies higher that 1 GHz, which allows us to run *K=1000* iterations in 1 *us* or to have a nanosecond reaction with fewer numbers of iterations.

One could think that the chosen number of steps within figures $N = 4$ ($N + 1 = 5$ dots) serves for illustrative purposes only, though we will show that it provides rather good approximation as well. Having a step *h=1/4=0.25*, the scheme (8)-(9) provides the accuracy of approximation $\varepsilon \approx h^4 = 0.25^4 \approx 0.004$, using the same value in formula (9), that requires about 20 iterations to run. Tang Nano 9k FPGA, having a modest cost of some ten pounds, can handle a mesh of $N = 10$, that provides a theoretical accuracy of computations of about 0.0001.



a) real numbers; b) integer approximation on FPGA.

Fig. 16. Dependence of the solution time (number of iterations) on the actual accuracy of approximation.

For computations represented with Fig. 16, we were using fine-tuning which includes, for the embedded implementation, adjusting the following set of parameters: *N*, *UNIT_VALUE*, *K*, *ROUND_EDGE*. We reach the required accuracy by increasing *N* and decreasing *UNIT_VALUE* that requires increasing *K* to achieve theoretical accuracy, though because of rounding error accumulation, it shifts the result with respect to the precise solution that is adjusted by the appropriate *ROUND_EDGE* choice.

As for the discussion of the presented approach prospects, we would like to note that in the present paper, we consider (for the case study) constant boundary conditions and, as a goal, the convergence to an unmovable or pulsing with low error configuration of the nonnegative integer approximation (a cellular automaton). For real-life embedded applications, extended systems are of certain interest where sensors are attached in the form of changing border conditions, and the goal values for optimal control are taken from certain specified locations of the lattice. In this case, we are not interested in halting the computing-communication structure but keeping it running and processing possible updates of sensors. For control systems that change time scale and have prolonged idling periods, switching to energy-saving

modes of functioning is possible when achieving good convergence to awake on a change of sensor inputs.

6 Conclusions

In this paper, we have applied infinite Sleptsov (generalized Petri) nets for computing-communication structure design aimed at fast mass-parallel numerical solving of practically significant problems specified by PDE or systems of PDE. The application domain includes big-scale simulations as a part of computer-aided design on general-purpose HPC architecture (supercomputers) and embedded applications with specialized hardware prototyped with FPGA. We studied systems having definite spatial structure in the form of a multidimensional square lattice; two-dimensional examples have been chosen for the case study.

We start with a given PDE or a system of PDEs and boundary conditions, then apply finite difference methods using a mesh of appropriate shape for its numerical solution with the required accuracy (error). We adjust the mesh to provide good convergence of the iterative solution process motivated by its ability for mass-parallel implementation. For an efficient solution, in an ideal case, the task spatial structure should be directly mapped into the corresponding computing-communication structure, which is possible for dedicated hardware implementation or its prototyping with FPGA for embedded control applications. For the HPC domain, we offer using general-purpose computing structures in the form of a multidimensional cube or torus with the corresponding topology of the communication subsystem. We focus on the connection mesh obtained using Moore's [21] or generalized neighborhood [22] for considerably better approximation (as a fourth degree compared to the second degree depending on the finite difference mesh step size).

Infinite models of a square lattice with Moore's neighborhood were constructed using the cut-through, store-and-forward, and combined switching and intermediate storage methods; infinite systems of linear algebraic equations were compiled and solved in parametric form; their invariance has been proven; properties of the model's conservativeness and boundedness have been proved. A future direction of research is the investigation of square lattices [14,22] with generalized neighborhood [23] by infinite nets.

Computing node SN programs have been designed to implement the iterative solution on the corresponding general-purpose computing-communication lattice. We presented a high-level approach based on the colored nets using real numbers in the process of computations, easily adjustable to using complex numbers as well.

For embedded applications, we considered the integer number approximation of the iteration technique that represents, in essence, a cellular automaton approach. The corresponding low-level specifications in the SN graphical language have been presented as an intermediate construct to generate Verilog code for dedicated hardware implementations or prototyping on FPGA. The corresponding FPGA implementation has been presented, benchmarks, and prospects of application for fast process control are discussed. It concerns plane and rocket engines, (thermo) nuclear power stations, the trajectory of hypersonic moving objects, etc.

Acknowledgements

The authors express their sincere gratitude to aseries of grants, establishments, and countries, that partially supported thisresearch, in particular: the JESH grant of the Austrian Academy of Science,2022; the PAUSE grant of the College de France, France, 2022-2023; the EU MCSA4UKRAINEgrant, 2023; the Hessen Land visiting professorship, Germany, 2023-2024; andthe Max Plank Institute for Software Systems visiting professorship, 2024.

References

1. C. L. Gardner, *Applied Numerical Methods for Partial Differential Equations* (Springer Nature Switzerland, 2024).
2. A. A. Samarskii, *Theory of Difference Schemes* (Taylor & Francis Group, 2001).
3. J. Strikwerda, *Finite Difference Schemes and Partial Differential Equations* (Second Edition, Society for Industrial Mathematics, 2007).
4. Bi, Zhuming Wang Xiaoqin, *Computer Aided Design and Manufacturing* (Wiley, 2020).
5. Thomas Sterling, Maciej Brodowicz, Matthew Anderson, *High Performance Computing: Modern Systems and Practices* (Morgan Kaufmann Publishers, 2024).
6. Adam Taylor, Dan Binnun, Saket Srivastava, *A Hands-On Guide to Designing Embedded Systems* (Artech House, 2021).
7. Y. Ajima et al., The Tofu Interconnect D, in *Proc. 2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Belfast, UK, 2018, 646-654.
8. T. Shimizu, Supercomputer Fugaku: Co-designed with application developers/researchers, in *Proc. 2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Hiroshima, Japan, 2020, 1-4.
9. Jack Dongarra, the 2021 ACM A.M. Turing Award recipient, discusses his work and accomplishments, https://youtu.be/Oe9LRKoE6L0
10. G. Goldrian et al., QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine, *Computing in Science & Engineering* **10**(6) (Nov.-Dec 2008) 46-54.
11. Hideharu Amano, ed., *Principles and Structures of FPGAs* (Springer, 2018).
12. Joseph Cavanagh, *Verilog HDL: Digital Design and Modeling* (CRC Press, 2017).
13. D. A. Zaitsev, Sleptsov Nets Run Fast, *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **46**(5) (2016) 682–693.
14. Dmitry A. Zaitsev, Ivan D. Zaitsev and Tatiana R. Shmeleva, Infinite Petri Nets: Part 1, Modeling Square Grid Structures, *Complex Systems* **26**(2) (2017) 157-195.
15. Dmitry A. Zaitsev, Ivan D. Zaitsev and Tatiana R. Shmeleva, Infinite Petri Nets: Part 2, Modeling Triangular, Hexagonal, Hypercube and Hypertorus Structures, *Complex Systems* **26**(4) (2017) 341-371.
16. A. I. Sleptsov, A. A. Yurasov, *Computer-Aided Design of Flexible Computer-Aided Manufacturing Systems* (Tekhnika Kiev, 1986).
17. K. Jensen, L. M. Kristensen, *Colored Petri Nets* (Springer Berlin, 2009).
18. Dmitry A. Zaitsev, Tatiana R. Shmeleva, Anatoly I. Sleptsov, Reenterable Colored Petri Net Models of Networks, Grids, and Clouds: Case Study for Provider Backbone Bridge, *in Proc. 26th Telecommunications Forum (TELFOR 2018)*, Belgrade, Serbia, November 20-21, 2018, Invited paper
19. D. A. Zaitsev, T. R. Shmeleva, Verification of hypercube communication structures via parametric Petri nets, *Cybernetics and Systems Analysis* **46**(1) (2010) 105-114.
20. Dmitry A. Zaitsev, Tatiana R. Shmeleva, Werner Retschitzegger, Spatial Specification of Grid Structures by Petri Nets, Lecture Notes in Networks and Systems, in *Proceedings of 4th ICMETE 2020, Micro-Electronics and Telecommunication Engineering*, 2021, 253-263.

21. J. Kari, Theory of cellular automata: a survey, *Theoret. Comput. Sci.* **334** (2005) 3–33.
22. D. A. Zaitsev, Simulating Cellular Automata by Infinite Petri Nets, *Journal of Cellular Automata* **13**(1-2) (2018) 121-144.
23. D. A. Zaitsev, A generalized neighborhood for cellular automata, *Theoretical Computer Science* **666** (2017) 21-35.
24. T. Atsue, E. V. Tikyaa, S. C. Nwokike, A numerical solution of the 2D Laplace's equation for the estimation of electric potential distribution, *Journal of Scientific and Engineering Research* **5** (2018) 268-276.
25. P. V. Patil, D. J. K. Prasad, Numerical solution for two-dimensional Laplace equation with Dirichlet boundary conditions, *IOSR Journal of Mathematics* **6**(4) (2013) 66-75.
26. B. Glushica, A. Kuhar, Toseva V. Arnautovski, Accuracy Investigation of FDM, FEM and MoM for a Numerical Solution of the 2D Laplace's Differential Equation for Electrostatic Problems, *The Journal of CIEES* **1**(2) (2021) 26–30.
27. D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth and T. Hoefler, An In-Depth Analysis of the Slingshot Interconnect, *in Proc. SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Atlanta, GA, USA, 2020, 1-14.
28. S. Gill, Parallel Programming, *The Computer Journal* **1**(1) (1958) 2–10.
29. C. A. Petri, Kommunication mit Automaten, PhD thesis, Technischen Hoschule Darmstadt (Germany, 1962) 128.
30. Tadao T. Murata, Petri Nets: Properties, Analysis and Applications, *Proc. of the IEEE* **77**(4) (1989) 541-580.
31. D. A. Zaitsev, J. Jürjens, Programming in the Sleptsov net language for systems control, *Advances in Mechanical Engineering* **8**(4) (2016) 1-11.
32. Dmitry A. Zaitsev, Strong Sleptsov nets are Turing complete, *Information Sciences* **621** (2023) 172-182.
33. D.A. Zaitsev, Universal Sleptsov Net, *International Journal of Computer Mathematics* **94**(12) (2017) 2396-2408.
34. Dmitry A. Zaitsev, Tatiana R. Shmeleva, Qing Zhang, and Hongfei Zhao, Virtual Machine and Integrated Developer Environment for Sleptsov Net Computing, *Parallel Processing Letters* **33**(3), 2350006 (2023).
35. A. A. Kostikov, N. D. Zaitsev, O. V. Subotin, Realisation of the double sweep method by using a Sleptsov net, *Int. J. Parallel, Emergent Distributed Systems* **36**(6) (2021) 516–534.
36. D. A. Zaitsev, *Clans of Petri Nets: Verification of Protocols and Performance Evaluation of Networks* (LAP Saarbrucken, 2013).
37. J. D. Ullman, *Elements of ML Programming* (2nd Edition Pearson, 1994).
38. Dmitry A. Zaitsev, Tatiana R. Shmeleva, Zeyu Zhou & Ding Liu, Verification of cryptocurrency consensus protocols: reenterable colored Petri net model design, *International Journal of Parallel, Emergent and Distributed Systems* **39(**1) (2024) 32-50.
39. Kazuhiro Ogata, Narciso Martí-Oliet, eds., *Rewriting Logic and Its Applications*, 15th International Workshop, WRLA 2024, Luxembourg City, Luxembourg, April 6-7, 2024, Revised Selected Papers, (Springer, 2024).
40. T. R. Shmeleva, D. A. Zaitsev, I. D. Zaitsev, Verification of square communication grid protocols via infinite Petri nets, in *Proc. 10th Middle Eastern Simulation Multiconference (MESM 2009)*, September 27-29, Beirut, Lebanon, 2009, 53-59.
41. Liberzon Daniel, *Switching in Systems and Control* (Published by Birkhäuser Boston, 2003).
42. G. Berthelot, R. Terrat, Petri Nets Theory for the Correctness of Protocols, *IEEE Transactions on Communications* **30**(12) (1982) 2497–2505.
43. D. A. Zaitsev, T. R. Shmeleva, Parametric Petri Net Model for Ethernet Performance and Qos Evaluation, in *Proc. of 16th Workshop on Algorithms and Tools for Petri Nets*, September 25-26, University of Karlsruhe, Germany, 2009, 15-28.

44. R. Baxter et al., Maxwell - a 64 FPGA Supercomputer, *in Proc. Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, Edinburgh, UK, 2007, 287-294.
45. D. A. Zaitsev, Compositional analysis of Petri nets, *Cybernetics and Systems Analysis* **42**(1) (2006) 126-136.
46. B. Berthomieu, O.-P. Ribet, F. Vernadat, The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets, *Int. J. Prod. Res.* **42**(4) (2004) 2741–2756.
47. D. A. Zaitsev, Paradigm of Computations on the Petri Nets, *Automation and Remote Control* **75**(8) (2014) 1369–1383.
48. Ruiyao Xu, Si Zhang, Ding Liu, and Dmitry A. Zaitsev, Sleptsov net based reliable embedded system design on microcontrollers and FPGAs, in *Proc. of 2024 IEEE International Conference on Embedded Software and Systems (ICESS)*, Dec. 13-15, Wuhan, China, 2024.